Information & Computing－4

オートマトン 言語理論 計算論 II

［第2版］

J.ホップクロフト／R.モトワニ／J.ウルマン＝共著
野崎昭弘／高橋正子／町田 元／山崎秀記＝共訳

INFORMATION & COMPUTING

サイエンス社

Information & Computing—4

オートマトン 言語理論 計算論 II ［第2版］

J.ホップクロフト／R.モトワニ／J.ウルマン＝共著
野崎昭弘／高橋正子／町田　元／山崎秀記＝共訳

サイエンス社

サイエンス社のホームページのご案内
http://www.saiensu.co.jp
ご意見・ご要望は　rikei@saiensu.co.jp　まで．

訳者まえがき

本書は，原著者の序文にも記されている通り，1969 年にホップクロフトとウルマンの共著として出版された『言語理論とオートマトン』(サイエンス社, 1971 年) と，1979 年に出版された『オートマトン・言語理論・計算論／I・II』(サイエンス社，1984 年・86 年) の全面改訂版である．学部の講義に合うように，深さを追求するよりは「基礎的な部分の直観的な理解を確実にする」ことがめざされていて，前著よりずっと説明が詳しくなっている．その分「風格が落ちた」とか「自分で考えるところが少なくなった」という批判もあろうが，これでもまだ「むずかしい」と感じる読者はおられるだろうから，これまでの著書に代わる，新しい教科書の一つの標準になることと予想される．

翻訳の分担は以下の通りである．

1, 5, 6 章と序文	… 野崎昭弘
2, 3 章	… 高橋正子
4, 10, 11 章	… 町田元
7, 8, 9 章	… 山崎秀記

なお分量の関係で，第 1 章から 7 章までを I 巻，そのあとを II 巻として出版することになった．訳語については，前著にあるものは基本的にそれに従い，新しいものは標準的な情報科学辞典などを参照して，なるべく一般的な訳語を使うように努めた．しかしそれでもいくつかの用語について，訳者の間で意見が分かれる場合もあった．たとえば本文中の induction はもちろん「(数学的) 帰納法」でよいが，証明の中に現れる Basis ($n=1$ の場合) と Induction (n から $n+1$ へ) については,「帰納的ステップ」,「帰納法のステップ」などを含む意見が対立し，結局野崎の判断 (というよりは趣味で)「基礎」,「帰納」という短い用語を当てることにした (本文中の inductive step は「帰納的ステップ」にしたので,「帰納」はその省略形とお考えいただきたい).

本訳書によって，これからも日本の情報科学の土台がしっかり維持されることを，訳者一同期待している．しかし何しろ大部の書物であるから，思わぬ見落としや不適切な翻訳が残っているかも知れない．お気づきの点は，どしどし

ご教示いただけると幸いである．

編集の段階で，サイエンス社編集部の田島伸彦さん，鈴木綾子さんには大変お世話になった．特に鈴木さんは原稿と校正刷りを綿密にチェックされ，訳者の見落とした不統一・訳し洩れ・誤字などを数多く指摘して下さった．この場を借りて深くお礼を申し上げたい．

2003 年 2 月 18 日

野崎昭弘
高橋正子
町田　元
山崎秀記

序　　文

1979 年に出た本書の前身では，序文の中でホップクロフトとウルマンはオートマトンの話題が，彼らの最初の本が出た 1969 年頃の状況にくらべて，爆発的に拡がったという事実に驚嘆している．実際，1979 年の本は前の著書になかった多くのトピックを含んでいて，大きさは約 2 倍に膨らんでいた．その 1979 年版と比べると，本書は 1970 年代の自動車のように，「外から見ると大きくて，中は小さく」なっている．これは逆戻りのように聞こえるかも知れないが，著者たちはいくつかの理由でこの変化を歓迎している．

まず 1979 年には，オートマトンと言語の理論は活発な研究領域であった．あの本の目的は数学的な傾向の学生がその分野で新しい貢献をするのを励ますことであった．今日，オートマトン理論を直接めざす研究は（応用とは対照的に）僅かで，1979 年版の簡潔で，高度に数学的なトーンを保つ理由はほとんどない．

第 2 に，オートマトンと言語の理論の役割は，過去 20 年間で変わった．1979 年には，オートマトンはおもに大学院レベルの科目で，我々は読者として進んだ大学院生を想定していた——後半の章を読む学生は，特にそうである．今では，この科目は学部カリキュラムの一つの基本科目である．そこで本書の内容は，学生の予備知識を少なくし，基礎的な事柄の説明を増やし，前著より詳しい議論を書かねばならなかった．

第 3 に環境の変化がある．CS（コンピュータ科学）が過去 20 年間に予想を超えて成長を遂げた．1979 年にはカリキュラムを「これから次の技術革新を越えて生き残る」と確信できる材料で満たすのは容易なことではなかったのに，今では非常に多くの話題・技法が，学部カリキュラムの限られた時間の中でせめぎあっている．

第 4 に，CS はより職業的な科目になり，多くの学生に厳しい実利主義が生まれている．我々はオートマトン理論の諸相が多くの新しい教科で本質的な道具であると今でも信じているし，学生がすぐにお金になる技術だけを習いたいとどれほど強く望んだとしても，オートマトン理論の標準的なコースに含まれている理論的で視野を拡げる練習問題は今でも価値を失っていない，と信じて

いる．しかし，コンピュータ科学の学生が選べる科目リストの中に確実に位置を占め続けるためには，数学と並んで応用を強調する必要がある，と思われる．そのため，前著のいくつかの抽象的な話題を現在そのアイデアがどのように使われているかの例示に置き換えた．オートマトンと言語の理論のコンパイラへの応用はよく知られていてふつうコンパイラのコースでカバーされるが，他にもプロトコルを検証するモデル検査アルゴリズム，文脈自由言語を模型とするドキュメント記述言語など，新しい応用がたくさんある．

本書で拡大と縮小が同時進行した最後の要因は，ドン・クヌースとレス・ランポートが開発した植字システム TEX と LATEX を利用できたことである．特に後者のお陰で，植字の「開かれた」方式が可能になり，本書が大きく，しかし読みやすくなった．我々はお二人の努力に感謝している．

この本の使用法

本書は 3 年生以上の半年コース，あるいはその半分のコースで使うのに適している．スタンフォードでは，オートマトンと言語理論の中のコース CS153 でこの本書の素稿を使ったが，これは 4 分の 1 年のコースで，Rajeev と Jeff の 2 人が教えた．時間が限られていたので，第 11 章は扱わず，こみいった多項式時間リダクションなど，後ろの方の材料のいくつかも省かれた．本書のウェブ・サイト（あとに示す）には，CS154 のための何回かの講義ノートとシラバスがのっている．

何年か前に，我々はスタンフォードに来た大学院生の多くが，オートマトン理論は習っているが「実際的に解けない」可能性の理論を習っていないことに気がついた．スタンフォードの教員はコンピュータ科学者なら誰でも「NP 完全だととても時間がかかる」程度では困るので，それ以上の理解を得るためにはその理論は欠かせないと考えているため，もう一つのコース CS154N が用意されている．そのコースでは学生は第 8，9，10 章を読めばよい．実際には CS154N のための予備知識としては，CS154 の最後の 3 分の 1 で，これらの章が役立つ．今日でも，各学期ごとにこの選択科目を活用する学生がいる．余分な努力はごくわずかですむので，この選択はおすすめしたい．

予 備 知 識

本書を利用する最もよい方法は，グラフ，木，論理，および証明技法などを含

む離散数学のコースを取っておくことである．プログラミングのコースや，一般的なデータ構造，再帰，コンパイラなどシステムの主要な構成要素についても学んでいることを仮定している．これらの基礎科目は，1〜2 年生用の標準的な CS プログラムで学べる．

演　習　問　題

本書はほとんどすべての節にいくつかずつ，幅広い練習問題をのせている．むずかしい問題（あるいはむずかしい部分）には感嘆符 **!** がついている．最高にむずかしい問題には二重の感嘆符 **!!** がついている．

星印 ***** がついた問題（あるいはその一部分）もある．そういう問題には，ウェブ・ページを通して解答を見られるように努力している．それらの解答は公開され，自習のために利用できる．場合によっては，ある問題 B を解くのに別の問題 A の解答の変形あるいは応用が必要になることに注意してほしい．問題 A のある部分が解けたなら，B の対応する部分も解けると思ってよい．

WWW によるサポート

本書のホームページは次の通り．

`http://www-db.stanford.edu/~ullman/ialc.html`

ここには星印つきの問題の解答，気がついた誤りの訂正，参考資料が掲載されている．我々は CS154 を教えるたびに，その講義ノートを（宿題・解答・試験問題も）利用できるようにしたいと思っている．

謝　　辞

「証明をどう書くか」という Craig Silverstein のプリントは，第 1 章の一部の材料として役だった．本書の素稿について，コメントと誤りの訂正が次の人々から寄せられた：Zoe Abrams, George Candea, Haowen Chen, Byong-Gun Chun, Jeffrey Shallit, Bret Taylor, Jason Townsend, Erik Uzureau
これらの人々に深く感謝する．残った誤りはもちろん我々の責任である．

ニューヨーク・イタカ，カリフォルニア・スタンフォード

2000 年 9 月　　J. E. H.　R. M.　J. D. U.

目　　次

9 決定不能性

10 実行不能な問題

11　その他の「問題のクラス」

[第 I 巻]

8 テューリング機械入門

この章では，議論の方向をおおきく変える．今までは，主に単純な言語クラスを調べてきた．その応用も，プロトコルの解析，テキスト探索，プログラムの構文解析など，比較的制約されたものであった．ここで，計算機構で定義できる言語全体のクラスが，どのようなものであるかという問題を調べる．実はこの問題は，コンピュータに何ができるか，という問題と同じである．なぜなら，ある言語に属す列を認識させることは，問題を形式的に表現する一つの方法であり，問題を解くことは，コンピュータがすることの代名詞であるから．

C 言語の知識を仮定して，コンピュータを使って解くことのできない特定の問題があることの直観的な議論から始めよう．このような問題は,「決定不能」と呼ばれる．次に，コンピュータの古典的な形式化であるテューリング (Turing) 機械を導入する．テューリング機械はパソコンとは似ても似つかないし，新会社がその生産・販売を決定することなどあり得ないだろう．しかし，テューリング機械は任意の物理的計算機構ができることの正確なモデルとして広く受け入れられている．

第 9 章において，テューリング機械を,「決定不能」問題，すなわちどのコンピュータも解けない問題に関する理論を展開するのに用い，簡単に表現できる多くの問題が決定不能であることを示す．一つの例は「与えれた文法があいまいか否か」であり，他にもたくさんある．

8.1 コンピュータで解けない問題

この節の目標は，特定の問題が計算可能でないことの証明に対する C プログラムに基づく直観的な導入を提供することである．議論する特定の問題というのは「ある C プログラムが最初に表示する文が hello, world か否か」である．プログラムをシミュレーションすれば，プログラムが何をするかわかる筈だと

思うかも知れない．しかし，現実には，何かを出力するまでに想像もできないほど長い時間を要するプログラムも扱わなければならない．この問題 —— 何かが起きるにしても何時起きるかわからない —— という問題は，プログラムの動作を予言することの不可能性の決定的な理由である．しかし，定められた仕事をこなすプログラムが存在しないことの形式的証明は極めて技巧的であり，形式的な仕組みを整えなければならない．この節では形式的証明の裏にある直観を与えよう．

8.1.1 "hello, world" と表示するプログラム

図 8.1 は，カーニハン (W. Kernighan) とリッチー (M. Ritchie) の古典的な本[1]を読む学生が最初に出会う C プログラムである．プログラム 8.1 が hello, world と表示して終了することは容易にわかる．このプログラムはきわめて平明で，以来，言語の入門に「その言語で hello, world と印刷するプログラムをどのように書くかを示す」ことが共通の習慣になった．

```
main()
{
    printf("hello, world\n");
}
```

図 8.1 カーニハンとリッチーの hello-world プログラム

しかし，hello, world と表示するプログラムは他にもある．図 8.2 は，自明とはとてもいえないけれども，hello, world と表示しそうな別のプログラムである．プログラム 8.2 は入力 n を受け取り，方程式 $x^n + y^n = z^n$ の正整数解を探す．方程式を満たす正整数 x, y, z が一つでも見つかれば，hello, world と表示し，見つからなければ，永久に探し続け，hello, world と表示することは決してない．

---

[1] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 1978, Prentice-Hall, Englewood Cliffs, N.J., 日本語版 プログラミング言語 C 第 2 版石田晴久訳 共立出版

```
int exp(int i, n)
/* computes i to the power n */
{
    int ans, j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    return(ans);
}

main ()
{
    int n, total, x, y, z;
    scanf("%d", &n);
    total = 3;
    while (1) {
        for (x=1; x<=total-2; x++)
            for (y=1; y<=total-x-1; y++) {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hello, world\n");
            }
        total++;
    }
}
```

図 8.2　hello-world プログラムとして表現されたフェルマー (Fermat) の最終定理

このプログラムの動作を理解するために，まず exp が指数乗を計算する補助関数であることに注意しよう．主プログラムは，三つ組 (x, y, z) を，任意の正整数の三つ組をいつかは調べることが確実であるような順番で探索する．このような探索を正しく行うために，第 4 の変数 total を使う．total は 3 から始めて，while ループの中で繰り返しの度に 1 ずつ増え，したがって，どんな正整数にもいつかは到達する．while ループの中では，total を次のようにして三つ

の正整数 x, y, z に分割する．まず，x を 1 から $total - 2$ まで変化させ，そのfor ループの中で，y を 1 から $total - x - 1$ まで変化させ，最後に total から x と y を引いた残りを z に割り当てる．

最も内側のループでは，組 (x, y, z) が等式 $x^n + y^n = z^n$ を満たすか否かが検査される．式を満たせば，プログラムは hello, world と表示し，満たなさければ，何も表示しない．

プログラムが読み込む n の値が 2 ならば，プログラムはいつかは $x^n + y^n = z^n$ を満たし，たとえば $total = 12$, $x = 3$, $y = 4$, $z = 5$ のような整数の組を見いだす．したがって，入力 2 に対しては，プログラムは確かに hello, world と表示する．

しかし，整数 $n > 2$ に対しては，プログラムが $x^n + y^n = z^n$ を満たす三つ組を見いだすことは決してなく，hello, world と表示することはない．興味深いことに，数年前までこのプログラムがある大きな n に対して hello, world と表示することがあるか否か，わからなかった．そうでないという主張，つまり $n > 2$ ならば $x^n + y^n = z^n$ は整数解を持たない，という主張は 300 年前にフェルマー (Fermat) によってなされたが，ごく最近までその証明は見つからなかった．この主張は「フェルマーの最終定理」と呼ばれる．

hello-world 問題を次のように定義しよう．それは，与えられた C プログラムが与えられた入力に対し，最初の 12 文字として hello, world と表示するか否かを決定する問題である．これ以降，プログラムが表示する最初の 12 文字が hello, world であるということを，単に，プログラムが hello, world と表示する，といういい方をする．

数学者が，簡単な 22 行のプログラムの問題を解くのに 300 年もかかったとすれば，与えられたプログラムが与えられた入力のもとで，hello, world と表示するか否かという一般的な問題は，真に難しいに違いないと思われる．実際，数学者がまだ解けていないどのような問題も，あるプログラムがある入力のもとで hello, world と表示するか，という形の問題に還元 (帰着) できる．したがって，任意のプログラム P と入力 I に対して，P に I を与えて走らせたとき hello, world と表示するか否か検査できるプログラムがもし書けたら，それは驚くべきことである．しかし，そんなプログラムは存在しないことをこれから証明する．

決定不能問題が存在する理由

ここで議論している「hello, world 問題」といった特定の問題が決定不能であることの証明は技巧的であるが，プログラミングを含むような任意のシステムにおいて，ほとんどすべての問題が決定不能でなければならない理由を理解するのは難しくない．“問題” というのは，実際には，列の言語への所属問題であることを思い出そう．2 個以上の記号を持つアルファベット上の言語の個数は可算ではない．すなわち，言語と自然数とを，1 対 1 に割り当てるような方法は存在しない．

一方，プログラムは，有限アルファベット (典型的には ASCII 符号) 上の有限列であるから，可算である．すなわち，それらを長さの順に並べ，同じ長さのものは辞書式順序で並べることができる．したがって，1 番目のプログラム，2 番目のプログラム等々，一般的には任意の自然数 i に対して，i 番目のプログラムといういいかたができる．

結局，問題の個数に対し，それより無限に少ないプログラムしかないことがわかる．言語をランダムに取り出せば，それは決定不能であることがほとんど確実である．ほとんどの問題が決定可能であるように思えるただ一つの理由は，ランダムな問題にはほとんど興味がないからである．むしろ，かなり単純でよく構成された問題が好まれ，実際，そのような問題は決定可能であることが多い．しかし，興味深く，明確かつ簡潔に述べることができる問題の中にも，決定不能な問題を多く見つけることができるのである．＜**訳注：**ランダムな問題は有限的に記述できるとも限らない．＞

8.1.2 仮想 “hello, world” 検査機

“hello, world” 検査機の作成不可能性の証明は，背理法である．すなわち，入力としてプログラム P と P の入力 I とを受け取り，P が hello, world と表示するか否か答えるプログラムが存在すると仮定する．それを H と呼ぼう．図 8.3 は H が行うことの説明である．特に，H の出力は 3 文字 yes か 2 文字 no のどちらかであり，必ずそのどちらかを出力する．

問題が，H のような，実例 (入力) に対して yes か no か正しく答えるアルゴリズムを持てば，その問題は「決定可能」であるという．さもなければ，その問題は「決定不能」である．ここでの目標は H が存在しないこと，すなわち，hello-world 問題が決定不能であることを証明することである．

命題を背理法で証明するために，H にいくつかの変更を加えて，最終的に存

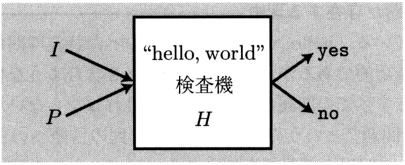


図 8.3　仮想 "hello-world" 検査機

在しないことを証明するプログラム H_2 を作る．H に加える変更は，任意の C プログラムに加えることのできる単純な変更なので，疑わしいのは H の存在だけであり，それは背理法のために仮定した命題そのものである．

議論を簡単にするために，考えている C プログラムについて 2，3 仮定する．これらの仮定は H の仕事を簡単にするだけで，難しくはしない．それゆえ，この制限された C プログラムに対する hello-world 検査器が存在しなければ，より広いクラスのプログラムに対する hello-world 検査器が存在しないのはあきらかである．仮定は

(1) すべての出力は文字を基本とし，グラフィックパッケージやその他の文字の形でない出力を行う機能を利用しない．

(2) すべての文字出力は関数 printf を利用し，putch やその他の文字出力関数を利用しない．＜**訳注：**(文字列) 変数の値の出力も禁止する．＞

さて，プログラム H が存在するとしよう．最初の変更は，出力の no を hello, world に変換することである．no は，H の入力であるプログラム P がその入力 I への応答として，最初に hello, world を出力することはないということを意味する，H の出力である．H が n を出力したとたんに，いつかはそれに o が続くことがわかる[2)]．

したがって，n (または no) を出力する printf 文を hello, world と出力するように変え，o だけを主力する printf 文は削除する．結果として，H_1 と呼ばれる新しいプログラムは，H と同様に動くが，H が no と出力するとき，そのときのみ，hello, world と出力する．H_1 を図 8.4 に示した．

プログラムの次の変換は，少し技巧的である．それは本質的に，テューリング

---

2) 普通，プログラムは一つの printf 関数で no と出力するだろうが，最初に n と出力した後，別の printf で o を出力するかもしれない．

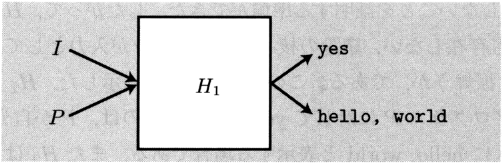


図 8.4 H_1 は H のように動くが，no の代わりに hello, world と出力する．

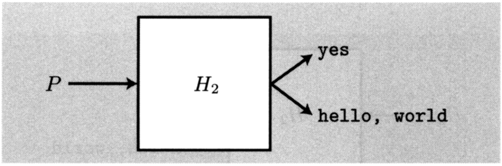


図 8.5 H_2 は H_1 と同様に振舞うが，入力 P を P と I の両方に使う．

がテューリング機械に関する非決定性定理を証明したときの直観でもある．実際に興味があるのは，他のプログラムを入力として受け取り，それについて何かを判定するプログラムなので，H_1 を次のように制限する．

(a) 入力を P と I ではなく，P だけにする．

(b) P が入力として P 自身のコードを受け取ったときに，どう動くか，すなわち,「H_1 は，入力プログラムとその入力 I として，ともに P を受け取ったときどう動くか」を問題にする．

図 8.5 に示したプログラム H_2 を作るために H_1 に以下の修正を加える．

(1) H_2 は最初に入力 P 全体を読み，配列 A に貯える．配列 A には，malloc で領域を割り当てる[3]．

(2) H_2 は次に H_1 を模倣する．しかし，H_1 がその入力 P や I を読むとき，H_2 は A に貯えられたコピーから読み取る．H_1 が入力 P と I をそれぞれどこまで読んだかを覚えておくために，H_2 は A の位置を指すカーソルを二つ使う．

---

[3] UNIX のシステム関数 malloc は，指定された大きさの記憶領域を割り当てる．この関数は，任意の長さの入力が読み込まれる場合のように，必要な記憶領域の量がプログラムを実行するまでわからないときに使われる．典型的には，入力がどんどん読まれ，それにつれて必要な記憶領域がどんどん大きくなるにつれて，malloc は何度も呼ばれる．

H_2 が存在しないことを証明する準備ができた．したがって，H_1 も存在せず，同様に，H も存在しない．議論の核心は，自分自身が入力として与えられたとき H_2 がどう振舞うか，である．この状況を図 8.6 に示した．H_2 が入力として与えられたプログラム P に対して yes と出力するのは，P が自分自身 P を入力されたときに hello, world と表示する場合である．また H_2 は，プログラム P が入力 P に対して最初に hello, world と出力しないとき，hello, world と出力する．

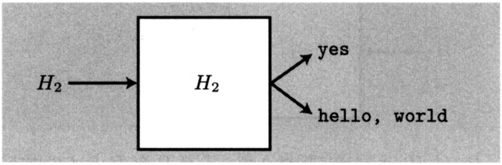


図 8.6　自分自身が入力として与えられたとき H_2 はどのように振舞うか．

図 8.6 の箱で表されたプログラム H_2 が yes と出力したとしよう．すると，箱 H_2 は，その入力 H_2 に対して，H_2 は入力 H_2 のもとでの最初の出力として hello, world と表示することを主張する．しかし，今仮定したことは，その状況で H_2 は yes と表示するということである．

したがって，図 8.6 の箱 H_2 の出力はもう一方の hello, world でなければならない．しかし，H_2 が自身を入力として与えられたときに最初に hello, world と表示するのなら，図 8.6 の箱の出力は yes でなければならない．すなわち，H_2 の出力としてどちらを仮定しても，もう一方が出力されることを結論できる．

この状況は自己矛盾であり，H_2 は存在しない．結局，H が存在するという仮定から矛盾が導かれたので，与えられたプログラム P とその入力 I に対し，P が hello, world と表示するか否か判定するプログラム H は存在しない．

8.1.3　別の問題への還元

どのコンピュータプログラムも解くことのできない一つの問題 —— 与えられたプログラムの与えられた入力に対する最初の出力が hello, world か否か答えよ —— が得られた．コンピュータに解くことのできない問題は**決定不能** (undecidable) であるという．決定不能の形式的な定義は 9.3 節で与えるが，今

は，この語を素朴な意味で用いる．ここで，別の問題がコンピュータで解けるかどうか決定したい，としよう．それを解くプログラムを書こうとしてもよい．しかし，解法がよくわからないとき，そのようなプログラムはもともと存在しないことを証明したくなるかも知れない．

おそらく，新しい問題に対して，hello-world 問題で行ったのと同じ技法で，決定不能性を証明できることもあるだろう．それを解くプログラムが存在すると仮定して，プログラム H_2 のような二つの相反する動作をする自己矛盾プログラムを開発できるだろう．しかし，決定不能であることがわかっている問題をすでに一つ知っているのだから，それを利用すれば，矛盾状況の存在を直接証明する必要はない．単に，新しい問題がもし解けたら，その解を利用してすでに決定不能であることがわかっている問題が解けることを示せば十分である．この戦略を図 8.7 に示した．この技法を問題 P_1 の問題 P_2 への**還元** (reduction) と呼ぶ．

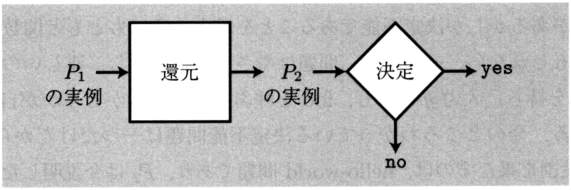


図 8.7　問題 P_2 が解ければ，その結果を使って問題 P_1 が解ける

問題 P_1 が決定不能だとわかっているものとしよう．そして，問題 P_2 を新たに，これも決定不能だと証明したいとする．そこで，図 8.7 の “決定” と書かれたひし形で表されるプログラムが存在すると仮定する．このプログラムは，問題 P_2 の実例がその問題の言語に属するか否かによって yes か no かを表示する[4)]．

問題 P_2 の決定不能性の証明のために，図 8.7 の正方形で示した “還元” を作り上げる必要がある．“還元” は，P_1 の実例を，同じ答を持つ P_2 の実例に作

4) 対象としている問題の実態は言語である．与えられたプログラムとその入力が最初に hello, world と表示するか否か決定する問題を議論したとき，実際に我々が議論したのは，C のソースプログラムの後に，そのプログラムが読む入力ファイルが続く文字列であった．この列の集合は，ASCII 文字上の言語である．

り変える．すなわち，言語 P_1 に属す任意の列は言語 P_2 に属す列に変換され，言語 P_1 に属さない任意の列は言語 P_2 に属さない任意の列に変換される．この“還元”が得られたら，P_1 を次のように解くことができる．

(1) P_1 の実例が与えられたら，すなわち，P_1 に属すか属さないかわからない語 w が与えられたら，“還元”アルゴリズムを適用して，列 x を生成する．

(2) x が P_2 に属すか否か決定し，w が P_1 に属すか否かと同じ答を得る．

$w \in P_1$ ならば，$x \in P_2$ であり，図 8.7 のアルゴリズムは yes と答える．$w \notin P_1$ ならば，$x \notin P_2$ であり，アルゴリズムは no と答え，どちらの場合も，w について正しく答える．列の P_1 所属問題を決定するアルゴリズムは存在しないのだから，背理法のために仮定した P_2 の決定アルゴリズムは存在しない．すなわち，P_2 は決定不能である．

例 8.1 この手法を使って，「プログラム Q が入力 y に対して，関数 foo を呼ぶことがあるか」が決定不能であることを示す．Q はもともと関数 foo を持たないかもしれない．その場合，問題はやさしい．しかし，難しいのは，Q が関数 foo を持ち，入力 y に対し，foo を呼ぶことになるかどうかが自明でない場合である．今のところわかっている決定不能問題は一つだけだから，図 8.7 で P_1 の役割を果たすのは，hello-world 問題であり，P_2 は今説明した calls-foo 問題である．calls-foo 問題を解く“決定”アルゴリズムがあると仮定しよう．課題は，hello-world 問題を calls-foo 問題に“還元”するアルゴリズムを設計することである．

すなわち，与えられたプログラム Q とその入力 y から，プログラム R とその入力 z を構成し，R が入力 z のもとで foo を呼ぶことと，Q が入力 y のもとで hello, world と表示することとが同値にならなければならない．その構成は難しくない．

(1) Q が関数 foo を持っていればその名前を付け換える，すなわち，関数名 foo と関数 foo のすべての呼び出しを新しい名前に書き換える．明らかに，新しいプログラム Q_1 は Q と同じ動作をする．

(2) Q_1 に関数 foo を付け加える．この関数は何もせず，呼ばれもしない．この結果を Q_2 とする．

コンピュータは本当にこれらのことができるのか?

図 8.2 のようなプログラムを調べる際,「フェルマーの最終定理の反例を本当に探索しているのか」を問題にすることができる．結局のところ，典型的なコンピュータでの整数は 32 ビット長であり，最小の反例が何兆という大きさの整数であれば，解が見つかる前にオーバーフローエラーが発生してしまうだろう．実際，128 メガバイトの主記憶と 30 ギガバイトのハードディスクを持つコンピュータは，たかだか $256^{30128000000}$ 個の状態しか持たず，それゆえ有限オートマトンにすぎないということもできる．

しかし，コンピュータを有限オートマトンとして扱うこと（あるいは有限オートマトンのアイデアの起源である脳を有限オートマトンとして扱うこと）は，生産的ではない．含まれる状態の数はとても大きく，その限界は明確ではないので，そのような扱いから有用な結論を引き出すことはできない．実際，コンピュータの状態数を好きなだけ拡張できると信じる理由は十分にある．

たとえば，整数を数字の任意の長さのリストとして表すことができる．もしメモリーを使い果たしてしまったら，プログラムは，今のディスクを外して横に置き新しい空のディスクと取り替えるよう人間に要求する文を表示できる．計算を進めるにつれて，コンピュータが必要とするだけの数のディスクの中で，何回でも取り替え要求を出すことができる．このプログラムは，図 8.2 のプログラムよりはるかに複雑であるが，我々のプログラミング能力を越えているわけではない．同様の技巧は，他のプログラムでも記憶領域の大きさや整数や他のデータ項目の大きさに関する，有限の限界を避けるのに使える．

(3)　Q_2 を変更して，それが表示する最初の 12 文字を大域的配列 A に貯える．結果を Q_3 とする．

(4)　Q_3 を次のように変更する．Q_3 が出力文を実行するときには，配列 A を調べて，すでに 12 文字以上出力したか否かを検査し，もしそうなら，最初の 12 文字が hello, world かどうかを調べ，hello, world なら，2 で付け加えた新しい関数 foo を呼ぶ．この結果がプログラム R で，R への入力 z は y と同じである．

Q が入力 y に対して，その最初の出力として hello, world と表示するとしよう．このとき構成された R は foo を呼ぶ．しかし，Q が入力 y に対して，最初の出力として，hello, world と表示しないならば，R は決して foo を呼ばない．R が入力 z に対し，foo を呼ぶか否かを決定できるとすれば，Q が入力 $y = z$

還元の向きは重要である

問題 P_2 が決定不能であることを示す際によくある誤りは,「もし P_1 が決定可能ならば, P_2 も決定可能である」という命題を証明して, P_2 を決定不能であることがわかっている問題 P_1 に還元しようとするものである. “P_1 が決定可能” という仮定が偽だから, その命題は確かに真であるが役に立たない.

新しい問題 P_2 が決定不能であることを証明する方法は, 知られている決定不能問題 P_1 を P_2 に還元することである. このやり方では「もし P_2 が決定可能ならば, P_1 も決定可能である」という命題を証明する. それは命題「もし P_1 が決定不能ならば, P_2 も決定不能である」の対偶であり, P_1 が決定不能であるので, P_2 も決定不能であると結論できる.

に対し hello, world と表示するか否かもわかる. hello, world 問題を解くアルゴリズムは存在しないことがわかっているし, R から Q を構成する四つのステップはプログラムのコード編集プログラムで実行できるから, calls-foo 検査器が存在するという仮定が間違っていたことになり, calls-foo 問題は決定不能である.

8.1.4 8.1 節の演習問題

♣ **問 8.1.1** hello-world 問題を以下の問題に還元せよ. 本節における素朴なスタイルを用いて適切なプログラム変換を記述し, 現実のコンピュータが持つファイルやメモリーの大きさの制限を考慮する必要はない.

*!a) 与えられたプログラムと入力に対し, そのプログラムがいつかは停止するか. すなわち, プログラムが入力に対し, 無限に動作するか否か.

b) 与えれたプログラムと入力に対し, プログラムが何らかの出力を出すか否か.

!c) 与えれた二つのプログラムと一つの入力に対し, それらのプログラムがその同じ入力のもとで同じ出力を出すか否か.

8.2 テューリング機械

決定不能問題の理論の目的は, そのような問題の存在を示すことそれ自身知的興味をかきたてるアイデアではあるが, それだけでなく, プログラマーにプログラミングを通して何が達成可能で何が達成可能でないかの指針を提供することである. この理論はまた, 第10章でやるように, 決定可能ではあるがそれ

を解くのに多大な時間を要する問題を議論する際に，より大きな実用的影響を持つ．このような問題は，“実行不能問題” と呼ばれ，決定不能問題が与えるより大きな困難を，プログラマーやシステムデザイナーに与える傾向がある．その理由は，決定不能問題は，通常そのことが明らかで，その解が実際に要求されるのはまれなのに対し，実行不能問題には日々直面する点にある．さらに，実行不能問題は，条件が少し変更された解や，ヒューリスティックな解を実際に求めることができたりすることが多い．したがって，デザイナーはきわめてしばしば問題が実行不能なクラスに属すか否か，そしてもしそうならどうすべきかの決定を迫られる事態に直面する．

問題が決定不能かあるいは実行不能かを証明するための道具が必要である．8.1 節で紹介した技法はプログラムに関する問題に対しては有効である．しかし，それを異なる分野の問題に変換するのはやさしくない．たとえば，hello-world 問題を文法があいまいか否かの判定問題に還元するのは難しいだろう．

そこで，C や他の言語のプログラムをもとにしたものでなく，テューリング機械と呼ばれる非常に単純な計算機モデルに基づくもので，決定不能性の理論を再構築しよう．テューリング機械は，本質的に，1 本の無限の長さの読み書き可能なテープを持った有限オートマトンである．計算可能の理論に関して，テューリング機械が通常の計算機より優れていることの一つは，テューリング機械が十分単純なので，その計算状況を正確に，PDA の ID と同様の単純な記法を使って，表せることである．それに対して，C プログラムの状態は，行われた関数呼び出しの列に現れるすべての変数 (の値) を含み，この状態を記述するための記法はあまりにも複雑すぎて，理解可能な形式的証明を作ることはほとんど不可能である．

テューリング機械の記法を用いて，プログラミングとは無関係に思える問題の決定不能性を証明する．例えば，9.4 節では文字列の組のリストに関する簡単な問題である「Post の対応問題」が決定不能であることを示す．そしてこの問題から容易に，文法のあいまい性などの判定が決定不能であることが示せる．同様に，実行不能問題を紹介する章では，計算とはあまり関係なさそうな問題 (たとえば論理関数の充足問題) が実行不能なことを示す．

8.2.1　すべての数学的問題を解くための旅

20 世紀への代わり目に，数学者ヒルベルト (D. Hilbert) は任意の数学的命題の真偽を判定するアルゴリズムを見つけることが可能かという問題を提出した．特に，彼は 1 階述語論理の任意の式を自然数上で真になるか否かを決定する方法を求めた．自然数上の 1 階述語論理は,「この文法はあいまいである」や「このプログラムは hello-world と表示する」といった命題を表すことができるので，ヒルベルトの企てが成功していたら，これらの問題が，すでに存在しないことがわかっている，決定アルゴリズムを持つことになる．

しかし，1931 年，ゲーデル (K. Gödel) は彼の有名な不完全性定理を発表した．彼は自然数上の述語論理の式で，その式自身述語論理の体系内で証明も反証もできないことが示される式を与えた．ゲーデルの技法は 8.1.2 項の自己矛盾プログラム H_2 の構成に似ているが，C プログラムではなく自然数上の関数を扱っている．

“可能な計算” というものに対して数学者が持つ概念は述語論理だけではない．実際，述語論理は計算的というより宣言的であり，他の多くの概念，—— 帰納的部分関数やよりプログラミング言語的な概念等々 —— と比べなければならない．1936 年にテューリングはテューリング機械を “可能な計算” のモデルとして提案した．このモデルはプログラムというより計算機に似ているが，真に電子的あるいは電気機械的な計算機が出現するのは，この数年後のことである (そしてテューリング自身，第 2 次世界大戦中にそのような機械の作成に携わった).

興味深いことに，計算のモデルの本格的な提案はすべて同じ計算能力を持つ．すなわち，それらは同じ関数を計算でき，同じ言語を認識できる．証明不可能な仮説，任意の一般的な方法で計算できるのは帰納的部分関数 (あるいは同じことだが，テューリング機械や現代のコンピュータが計算できる関数) だけであるという仮説は，論理学者チャーチ (A. Church) に因んで**チャーチの仮説** (Church’s Thesis) あるいは**チャーチ-テューリングの仮説** (Church-Turing Thesis) と呼ばれる．

8.2.2　テューリング機械の記法

テューリング機械は図 8.8 のように図示できる．この機械は有限個の状態の

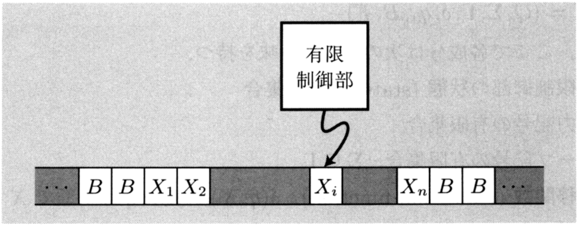


図 8.8 テューリング機械

一つをとり得る**有限制御部** (finite control) を持つ．**マス** (cell) に分けられた**テープ** (tape) があり，各マスには有限個の記号のどれかが書かれる．

最初，**入力アルファベット** (input alphabet) から選ばれた有限の長さの記号列が**入力** (input) としてテープ上に置かれる．テープの他のマスは左右に無限に続いており，そこには**空白** (blank) と呼ばれる特別の記号が書かれている．空白は**テープ記号** (tape symbol) であるが入力記号ではなく，テープ記号は入力記号や空白以外にも存在する．

常にテープ上のマスのどれかに位置する**テープヘッド** (tape head) があり，テューリング機械はそのマスを**読んでいる** (scanning) という．最初，テープヘッドは入力列が書かれている一番左のマスを読んでいる．

テューリング機械の**動作** (move) は有限制御部の状態と読んでいる記号の関数である．1 回の動作で，テューリング機械は次の動作を行う．

(1) 状態を変える．動作後の状態は現在の状態と同じであってもよい．

(2) 読んでいるマスにテープ記号を上書きする．上書きする記号はもとの記号と同じであってもよい．

(3) テープヘッドを左または右へ移動する．今の場合，ヘッドは動かなければならず停止させておくことはできない．この制限はテューリング機械の計算能力をそぐわけではなく，ヘッドを動かさないような動作の列は，それに続くヘッド移動を伴う動作と一緒に一つの動作にまとめることができる．

テューリング機械 (Turing machine, TM) の形式的な記法は，有限オートマトンや PDA と同様に，七つ組

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

を用いる．ここで各成分は次のような意味を持つ．

Q：有限制御部の**状態** (state) の有限集合

Σ：入力記号の有限集合

Γ：テープ記号の有限集合．$\Sigma \subseteq \Gamma$．

δ：**遷移関数** (transition function)．$\delta(q, X)$ の変数 q は状態で X はテープ記号である．$\delta(q, X)$ の値は，もし定義されていれば，(p, Y, D) で

(1)　p は Q に属する動作後の状態,

(2)　Y は Γ に属す記号で，読んでいるマスに上書きされる記号,

(3)　D は，それぞれ左または右を意味する記号 L または R で，ヘッドが動く方向を表す．

q_0：**初期状態** (initial state) で有限制御部が最初にとる状態を表す Q の要素．

B：空白記号．B は $\Gamma - \Sigma$ の元で，最初，入力記号を持つ有限個のマス以外のすべてのマスに置かれる．

F：**受理** (accepting) あるいは**最終** (final) 状態の集合で，Q の部分集合．

8.2.3 テューリング機械の時点表示

テューリング機械の動作を形式的に記述するために，PDA と同様の，**時点表示** (instantaneous description, ID) と呼ばれる計算状況を表す記法を用意しよう．TM は，原理的には無限に長いテープを持つため，TM の状況を簡潔に表すのは不可能に思うかも知れない．しかし，TM は有限回の動作の間に，その限界はいくらでも大きくなるにしても，有限個のマスしか訪れることはできない．したがって，すべての ID において，決して訪問されず入力も置かれなかった無限長の接頭部や接尾部があり，これらのマスにはすべて空白が置かれている．したがって，ID においては，一番左の非空白記号から一番右の非空白記号までのマスを示すことにする．ただし，ヘッドが前後につく空白の一つを読んでいるという特別な場合には，テープの非空白部の前後の空白を ID に含めなければならない．

ID では，テープに加えて，有限制御部の状態とヘッド位置を表さなければならない．そのために，状態をテープに埋め込み，読んでいるマスのすぐ左に置く．テープに状態を加えた列のあいまいさをなくすために，テープ記号として

使われている記号を状態として使わないように，$\Gamma \cap Q = \varnothing$ としなければならないが，TM の動作は状態がどのように呼ばれるかとは無関係なので，状態の名前を変えて，テープ記号と共通しないようにすることはやさしい．したがって，列 $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$ で次のような ID を表す．

(1)　q はテューリング機械の状態である．

(2)　テープヘッドは左から i 番目のマスを読んでいる．

(3)　$X_1X_2\cdots X_n$ は一番左の非空白記号から一番右の非空白記号までの部分である．例外として，ヘッドが一番左の非空白記号のさらに左にあったり，一番右の非空白記号のさらに右にある場合，$X_1X_2\cdots X_n$ の接頭部や接尾部が空白記号になり，i はそれぞれ 1 または n である．

テューリング機械 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ の動作を PDA で使った $\vdash_M$ 記法を使って表す．文脈から TM M が明らかなときは，1 回の動作を表すのに単に $\vdash$ を使う．同様に，$\overset{*}{\vdash}_M$ や $\overset{*}{\vdash}$ を用いて，TM M の 0 回以上の動作を示す．

$\delta(q, X_i) = (p, Y, L)$，すなわちヘッドが左に動くとする．このとき，

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \vdash_M X_1X_2\cdots X_{i-2}pX_{i-1}YX_{i+1}\cdots X_n$$

である．この動作で状態は p になりヘッドは左に動いてマス $i-1$ に位置する．二つの重要な例外がある．

(1)　$i = 1$ のとき，M は X_1 の左の空白マスに動く．この場合，

$$qX_1X_2\cdots X_n \vdash_M pBYX_2\cdots X_n$$

(2)　$i = n$ で $Y = B$ のとき，X_n に上書きされた空白 B は，後ろに続く無限の空白列と一緒になり次の ID には現れない．したがって，

$$X_1X_2\cdots X_{n-1}qX_n \vdash_M X_1X_2\cdots X_{n-2}pX_{n-1}$$

次に $\delta(q, X_i) = (p, Y, R)$，すなわちヘッドが右に動くとする．このとき，

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \vdash_M X_1X_2\cdots X_{i-2}X_{i-1}YpX_{i+1}\cdots X_n$$

である．この動作でヘッドは右に動いてマス $i+1$ に位置する．再び，二つの重要な例外がある．

(1)　$i = n$ のとき，X_{i+1} は空白で，もとの ID には現れない．したがって，

$$X_1X_2\cdots X_{n-1}qX_n \vdash_M X_1X_2\cdots X_{n-1}YpB$$

(2)　$i = 1$ で $Y = B$ のとき，X_1 に上書きされた空白 B は，前にある無限の空白列と一緒になり次の ID には現れない．したがって，

$$qX_1X_2\cdots X_n \vdash_M pX_2\cdots X_n$$

例 8.2　テューリング機械を設計して，それが典型的な入力列に対してどのように振舞うかを見てみよう．構成する TM は言語 $\{0^n1^n \mid n \geq 1\}$ を受理する．最初 0, 1 の有限列がテープ上に置かれ，その前後に無限長の空白が並ぶ．TM は，0 を X に 1 を Y に，交互に置き換えながら，その個数を照合する．

より詳しくは，入力の左端から出発して，以下を繰り返す：

> 0 があればそれを X に書き換え，1 を見つけるまで 0 や Y を読み飛ばして右に移動する．次に 1 を Y に書き換え，X を見つけるまで Y や 0 を読み飛ばして左に戻り，見つけた X のすぐ右に移動する．

入力が $\mathbf{0}^*\mathbf{1}^*$ に属さなければ，TM はいつか次の動作を行うことができなくなり，入力を受理することなく停止する．しかし，最後の 1 を Y に書き換えたその回で，0 をすべて X に書き換えていれば，入力は 0^n1^n の形をしていたことになり，受理される．TM M の形式的な記述は

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

で，δ は図 8.9 の表で与えられる．

状態	記号				
	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

図 8.9　$\{0^n1^n \mid n \geq 1\}$ を受理するテューリング機械

M が計算を進める途中で，M のテープは，正則表現 $\mathbf{X}^*\mathbf{0}^*\mathbf{Y}^*\mathbf{1}^*$ によって記述される記号列を持つ．すなわち，0 から書き換えられた X の列に続いてまだ書き換えられていない 0 が続き，その後，1 から書き換えられた Y の列とまだ書き換えられていない 1 の列が続く．後ろに続く 0, 1 の列は空列のこともある．

状態 q_0 は初期状態であり，M は残っている中で最も左の 0 に戻ったときにも状態 q_0 に入る．M は状態 q_0 で 0 を読んだとき，図 8.9 の左上隅にある規則により，状態 q_1 に遷移し 0 を X に書き換えて右に移動する．状態 q_1 に入ると，M は状態 q_1 に留まったままテープ上のすべての 0 と Y をを読み飛ばして右へ動く．M は X か B を見ると停止する．しかし，状態 q_1 で 1 を読むと，そ

の 1 を Y に書き換え，状態 q_2 に遷移し左に動き始める．

状態 q_2 では，M は 0 や Y を読み飛ばして，q_2 に留まったまま左に移動する．最も右の X に出会うと，それはすでに X に置き換えられた 0 のブロックの右端を表すので，状態 q_0 に戻り，右のマスへ移動する．ここで二つの場合が考えられる．

(1) 0 を読むと，すでに述べた照合サイクルを繰り返す．

(2) Y を読むと，それはすべての 0 を X に置き換えたことを意味する．この時点ですべての 1 が Y に置き換えられていれば，入力は 0^n1^n の形をしており，受理すべきである．したがって，M は状態 q_3 に入り，Y を読み飛ばして右に移動する．最初に見る Y 以外の記号が空白ならば，実際に同数の 0 と 1 があったことになるので，M は状態 q_4 に入り，入力を受理する．そうでなくて M がさらなる 1 に出会う場合，これは 1 の個数が多すぎることを意味するので，M は入力を受理することなく停止する．M が 0 に出会う場合は，入力は間違った形をしていることになり，やはり M は停止する．

M の受理計算例を示す．入力は 0011 である．最初，M は状態 q_0 にあり左端の 0 を見ている．すなわち，ID は $q_0 0011$ である．M の 0011 に対する動作列の全体は次のようになる．

$$
\begin{aligned}
& q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1 \\
& \vdash Xq_0 0Y1 \vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY \\
& \vdash XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 B \vdash XXYYBq_4 B
\end{aligned}
$$

別の例として，受理言語に属さない入力 0010 に対する動作列は次のようになる．

$$
\begin{aligned}
& q_0 0010 \vdash Xq_1 010 \vdash X0q_1 10 \vdash Xq_2 0Y0 \vdash q_2 X0Y0 \\
& \vdash Xq_0 0Y0 \vdash XXq_1 Y0 \vdash XXYq_1 0 \vdash XXY0q_1 B
\end{aligned}
$$

0010 に対する動作列は 0011 に対する動作列と，最後の 0 を見つける ID $XXYq_1 0$ までは同じである．ここで M は状態 q_1 に留まったまま右に動いて B を読み，ID $XXY0q_1 B$ に達する．しかし M は状態 q_1 で B に対する動作を持たず，したがって，入力を受理することなく停止する． ▨

8.2.4 テューリング機械の遷移図

テューリング機械の遷移関数は，PDA と同じように図示できる．遷移図は TM の状態を表す節点を含み，状態 q から p への矢印は，1 個以上の X/YD の形の項でラベル付けされる．ここで，X, Y はテープ記号で D はヘッドの移動方向である．すなわち，$\delta(q, X) = (p, Y, D)$ のとき，q から p への矢印にラベル X/YD を付ける．しかし，図では左 L を←，右 R を→で図案化した．

他の種類の遷移図と同様，開始 (初期) 状態を語，開始と状態へ向かう矢印とで示し，受理状態を 2 重丸で示す．したがって，遷移図から直接読み取れない TM の情報は，空白に用いられる記号だけであるが，特に指定しない限り，記号 B が使われるものと仮定する．

例 8.3　図 8.10 は例 8.2 のテューリング機械の遷移図である．その遷移関数は図 8.9 に与えた．▨

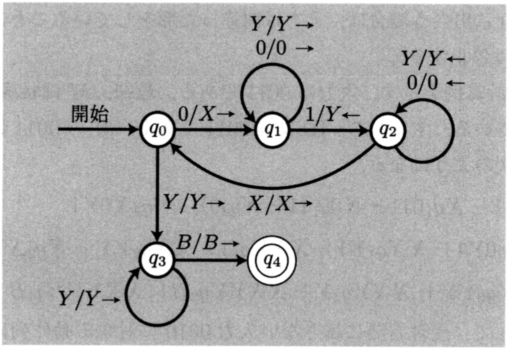


図 8.10　0^n1^n の形の列を受理する TM の遷移図

例 8.4　今日では，TM を言語の認識装置，あるいは同じことだが，問題解決装置とみなすのが一般的であるけれども，テューリングはもともと，TM を自然数関数の計算機として考えていた．彼の枠組みでは，自然数は 1 進数，つまり 1 種類の文字のブロックで表され，TM はそのブロックの長さを変えることや，別の場所に新しいブロックを構成することで計算を行う．ここで与える単純な例では，テューリング機械が差 (minus, proper subsruction) と呼ばれ

る，$m \dot{-} n = \max(m-n, 0)$ で定義される関数 $\dot{-}$ を計算する．すなわち，$m \dot{-} n$ は $m > n$ なら $m-n$，$m \leq n$ なら 0 である．

この演算を行う TM は

$$M = (\{q_0, q_1, \ldots, q_6\},\ \{0,1\},\ \{0,1,B\},\ \delta,\ q_0,\ B)$$

で与えられる．この TM は入力を受理するために使われるものではないので，7 番目の成分である受理状態の集合は省略した．M は前後に空白のついた 0^m10^n のテープ上で出発し，空白で囲まれた $0^{m \dot{-} n}$ をテープに残して停止する．

M は以下を繰り返す．まず，残っている最も左の 0 を見つけてそれを空白に置き換える．次に右に動いて 1 を探し，見つけたらさらに右に動いて最初に見つけた 0 を 1 に書き換える．その後，M は左に戻って最も左にある 0 を探すために，B に出会うまで左に動いた後 1 マス右に動く．この繰り返しが終わるのは，

(1) 0 を探して右に動いている間に B に出会う場合：0^m10^n 中の後半の n 個の 0 がすべて 1 に書き換えられ，最初の $n+1$ 個の 0 が B に書き換えられたことを意味する．M は $n+1$ 個の 1 を n 個の B と 1 個の 0 に書き換えて，$m-n$ 個の 0 をテープに残す．この場合，$m > n$ なので，$m \dot{-} n = m - n$ である．

(2) 繰り返しの最初で，M が空白 B に書き換えるべき 0 を見つけられなかった場合：これは最初の m 個の 0 がすべてすでに B に書き換えられているためで，$n \geq m$ で $m \dot{-} n = 0$ である．M は残っているすべての 1 と 0 を B で置き換え，完全に空白だけのテープを残して停止する．

図 8.11 に遷移関数 δ の規則を示した．図 8.12 の遷移図でも δ を与えている．以下は 7 個の状態それぞれの役割を説明しよう．

q_0：この状態では，0 を読むと繰り返しサイクルを開始させ，B を読むと繰り返しを停止させる．0 を読んだときは，0 を B に置き換え，ヘッドを右に移動させ，状態 q_1 に入る．一方，B を読んだときは，二つの 0 のグループの間の照合が終わっているので，状態 q_5 に遷移してテープ全体を空白で埋める．

q_1：この状態では，M は右に動いて，最初の 0 のブロックを通過し，最も左にある 1 を探す．1 が見つかったら状態 q_2 に入る．

q_2：M は右に動き，0 を見つけるまで 1 を読み飛ばす．見つけた 0 を 1 に変

状態	記号		
	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	−
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	−
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_3, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, L)	(q_6, B, R)
q_6	−	−	−

図 8.11 差 $\dot{-}$ を計算するテューリング機械

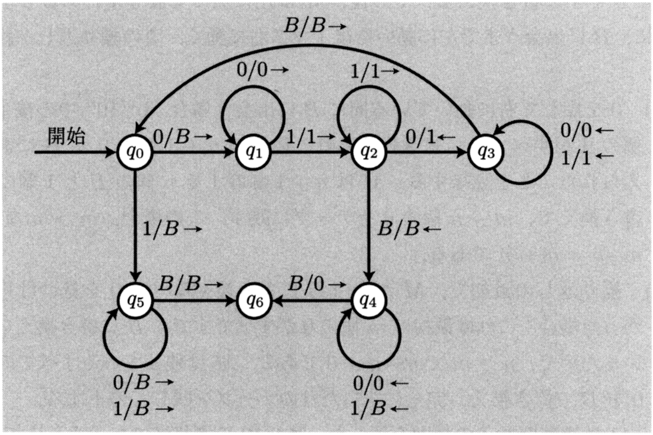


図 8.12 例 8.4 の TM の遷移図

え左に戻って状態 q_3 に入る．しかし，1 のブロックの右にもう 0 が残っていない場合には，M は空白に出会う．これは上に述べた (1) の場合にあたり，2 番目のブロックの n 個の 0 が，1 番目のブロックの m 個の 0 のうちの n 個を消すのに使われ，差が完成している．M は，1 を空白に変換するために状態 q_4 に入る．

q_3：M は 0 と 1 を読み飛ばして空白を見つけるまで左に動く．B を見つけると，右へ 1 マス動いて状態 q_0 に戻り，サイクルが再び始まる．

q_4：ここで，差が完成する．しかし，最初のブロックの 0 のうち一つだけが対応する 0 のないまま B に変わっている．それゆえ，M は 1 を B で置き換えながらテープ上の B に出会うまで左に動き，見つけた B を 0 に戻し，状態 q_0 に入って停止する．

q_5：状態 q_5 には状態 q_0 から，最初のブロックのすべての 0 が B に書き換えられたことがわかったときに入る．この場合は 21 ページの (2) にあたり，差の結果は 0 である．M は残ったすべての 0 と 1 を B に変えて状態 q_6 に入る．

q_6：この状態の目的のすべては，仕事を終えた後で M を停止させることである．差がもっと複雑な別の関数のサブルーチンであるとき，q_6 はこれを使う計算の次のステップを開始させる． ▨

8.2.5 テューリング機械の受理言語

これまでテューリング機械が言語を受理する方法を直観的に述べてきた．入力列はテープ上に置かれ，テープヘッドはその左端から出発する．TM がいつか受理状態に入れば，入力は受理され，そうでなければ受理されない．

より形式的には，$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ をテューリング機械とするとき，$L(M)$ はある受理状態 $p \in F$ とテープ記号列 α, β が存在して，$q_0 w \overset{*}{\vdash} \alpha p \beta$ となる $w \in \Sigma^*$ の集合である．この定義は，例 8.2 で，$0^n 1^n$ の形の列を受理するテューリング機械を議論したときにも使った．

テューリング機械を使って受理できる言語はしばしば**帰納的可算言語** (recur-

テューリング機械の記述の便法

テューリング機械に対して通常使われる記号は今まで見てきた他の種類のオートマトンと類似している．

(1) アルファベットの最初のほうの小文字は入力記号を表す．

(2) 大文字，典型的には終わりのほうの大文字は，テープ記号を表し，入力記号のこともある．しかし，B は一般に空白記号に用いる．

(3) アルファベットの最後のほうの小文字は入力記号列を表す．

(4) ギリシャ文字はテープ記号列を表す．

(5) 文字 q, p やそのあたりの小文字は状態を表す．

sively emumerable language, RE 言語) と呼ばれる.「帰納的可算」という術語は，テューリング機械以前の計算の形式化に基づくものであるが，言語や算術関数でテューリング機械と同じクラスを定義する．この術語の由来については，9.2.1 項の箱書きで議論する．

8.2.6 テューリング機械と停止性

テューリング機械で共通に使われるもう一つの受理の概念がある．停止による受理である．TM が**停止** (halt) するのは，状態 q に入り入力記号 X を読んでいる状況で，動作がない，すなわち，$\delta(q, X)$ が定義されていない場合である．

例 8.5　例 8.4 のテューリング機械 M は言語を受理するようには定義されてなく，むしろ算術関数を計算するものとして見ている．しかし，M は任意の 0 と 1 の列上で停止する．M はテープ上にどんな 0,1 の列があったとしても，0 の 1 番目のグループに対して，2 番目のグループを (もしあれば) すべて消し，状態 q_0 に到達して停止する．　▨

入力を受理するときには TM が停止すると常に仮定することができる．すなわち，受理される言語を変えることなく，受理状態 q に対しすべての $\delta(q, x)$ を未定義にできる．一般に，特に述べることなく

- 受理状態では TM は停止する

と仮定する．残念ながら，入力を受理しないときにも必ず停止することを常に要求することはできない．受理，非受理に関わらずいつかは停止するテューリング機械を持つ言語は**帰納的** (recursive) と呼ばれる．帰納的言語の重要な性質については 9.2.1 項から考え始めるが，与えられた問題を解くアルゴリズムがあれば，その問題は "決定可能" であるという．それゆえ，常に停止する TM は 9 章の決定可能性理論に重要な関わりがある．

8.2.7 8.2 節の演習問題

♣　**問 8.2.1**　次の入力に対する図 8.9 の TM の ID 列を示せ．

*a)　00

b)　000111

c)　00111

♣ **!問 8.2.2**　次の言語を受理する TM を設計せよ．

***a)**　同数の 0 と 1 を含む列の集合

b)　$\{a^n b^n c^n \mid n \geq 1\}$

c)　$\{ww^R \mid w$ は $0, 1$ の任意の列$\}$

♣ **問 8.2.3**　2 進法で，整数 N を入力としてとりそれに 1 を加える TM を設計せよ．正確にいうと，テープには最初，記号 \$ の後に 2 進数の N が置かれている．テープヘッドは \$ を読んでいて，状態は q_0 である．設計する TM は，テープ上に書かれた 2 進数の $N+1$ の一番左の記号を読んでいる状況で状態 q_f に入り停止する．$N+1$ を作るときに，必要なら記号 \$ を壊してもよい．たとえば $q_0\$10011 \overset{*}{\vdash} \$q_f10100$, $q_0\$11111 \overset{*}{\vdash} q_f100000$ となる．

a)　テューリング機械の遷移関数を求め，各状態の直観的意味合いを説明せよ．

b)　設計した TM の入力 \$111 に対する ID 列を示せ．

♣***! 問 8.2.4**　この問では，TM による関数計算と言語認識の同値性を示す．簡単のために，非負整数から非負整数への関数だけを考えるが，この問題のアイデアは任意の計算可能関数に適用できる．以下に二つの中心的な定義がある．

- 関数 f の**グラフ** (graph) を，$[x, f(x)]$ の形の列全体で定義する．ただし，x は 2 進の非負整数で，$f(x)$ は変数 x に対する関数 f の値を 2 進数で表した列である．
- テューリング機械が関数 f を**計算** (compute) するというのは，非負整数 x の 2 進表示をテープ上において開始したとき，$f(x)$ の 2 進表示をテープ上に書いて，停止することである．

次の問に，非形式的にしかし明確に答えよ．

a)　f を計算する TM が与えられたとき，f のグラフを受理する TM をどのように構成するか示せ．

b)　f のグラフを受理する TM が与えられたとき，f を計算する TM をどのように構成するか示せ．

c)　関数 f が**部分関数** (partial function) であるというのは，変数値に対して関数値が定義されなくてもよいときである．この問題のアイデアを部分関数に拡張するとき，$f(x)$ が定義されないような x に対しては，f を計算する TM は停止しなくてもよいものとする．(a) と (b) の読者の答は f が部分関数のときもうまく働くか．もしうまくいかないなら，どのように変更すればよいか．

♣ **問 8.2.5**　テューリング機械

$$M = (\{q_0, q_1, q_2, q_f\},\ \{0, 1\},\ \{0, 1, B\},\ \delta,\ q_0,\ B,\ \{q_f\})$$

を考える．δ が次の規則から成るとき，受理言語 $L(M)$ を非形式的にしかし明確に述べよ．

***a)**　$\delta(q_0, 0) = (q_1, 1, R); \delta(q_1, 1) = (q_0, 0, R); \delta(q_1, B) = (q_f, B, R)$.

b) $\delta(q_0, 0) = (q_0, B, R); \delta(q_0, 1) = (q_1, B, R); \delta(q_1, 1) = (q_1, B, R);$
$\delta(q_1, B) = (q_f, B, R)$.

!c) $\delta(q_0, 0) = (q_1, 1, R); \delta(q_1, 1) = (q_2, 0, L); \delta(q_2, 1) = (q_0, 1, R);$
$\delta(q_1, B) = (q_f, B, R)$.

8.3 テューリング機械のプログラム技法

この節の目的は，通常のコンピュータと似たやり方で，テューリング機械を計算に使う方法の感覚を得ることである．最終的には，TM は通常のコンピュータと同等の能力を持つことを納得してもらいたい．特に，テューリング機械は，他のテューリング機械に対して，8.1.2 項で示したような計算ができることを学ぶ．テューリング機械とコンピュータプログラムの両方にあるこの「自己適用」能力は，問題の決定不能性の証明を可能にする．

TM が持つ能力を明確にするために，テューリング機械の有限制御部やテープをどのように利用すればよいかを示唆する多くの例題を提供する．これらの技巧はどれも，TM の基本モデルの拡張ではなく，単に記法上の便法にすぎない．後に，その技巧を使って，たとえば，2 本以上のテープといった追加された機能を持つ拡張テューリング機械を，基本の TM で模倣する．

8.3.1 状態への記憶

有限制御部は，テューリング機械の “プログラム” 上の位置を記録するだけでなく，有限の大きさのデータを保持するのに使うことができる．図 8.13 にこの技法 (と次節で議論する多重トラックのアイデア) を示す．有限制御部は “制御” のための状態 q だけでなく，三つのデータ A, B, C を持つ．この技法は TM モデルの拡張を必要とせず，単に，状態をデータの組と考えるだけでよい．例 8.13 の場合には，状態を組 $[q, A, B, C]$ と考える．状態をこのようにみなすことによって遷移関数をより体系的に記述でき，TM プログラムに隠された戦略をより明確に表すことができる．

例 8.6 次の TM を設計する．

$$M = (Q,\ \{0,1\},\ \{0,1,B\},\ \delta,\ [q_0, B],\ B,\ \{[q_1, B]\})$$

M は最初に読んだ記号 (0 または 1) を有限制御部に記憶し，それが入力列のどこにも現れない，ということを検査する．したがって，M は言語 $\mathbf{01}^* + \mathbf{10}^*$ を

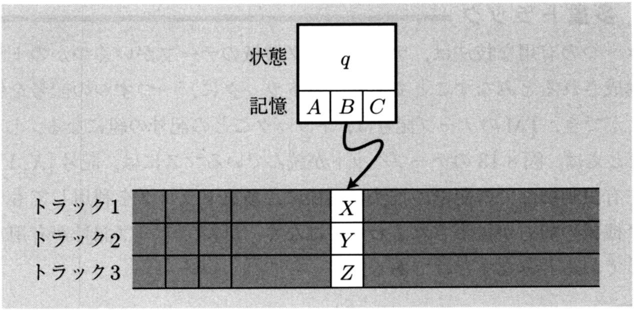


図 8.13 有限制御部の記憶装置と多重トラックを持つテューリング機械

受理する．このような正則言語の受理は，テューリング機械の能力を強調するものではないが，簡単な例証にはなる．状態の集合は $Q = \{q_0, q_1\} \times \{0, 1, B\}$ である．すなわち，状態は二つの成分の対と考えられる．

(a) 制御成分 q_0 または q_1 は，TM が今していることを表す．制御状態 q_0 は M がまだ列の最初の記号を読んでいないことを表し，q_1 は M がすでに最初の記号を読んでおり，空白記号に出会うことを期待しながら右に移動して，それがどこかに現れていないかを検査中であることを示す．

(b) データ成分は，最初に読んだ記号 0 または 1 を覚えて置くためのものである．この成分中の B は，記号がまだ読まれていないことを意味する．

M の遷移関数 δ は次の通りである．

(1) $a = 0, 1$ に対し，$\delta([q_0, B], a) = ([q_1, a], a, R)$．初期状態の制御部分は q_0 でデータ部分は B である．読まれた記号は，状態の第 2 成分にコピーされ，M は右に動き，制御状態 q_1 に遷移する．

(2) $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$．ここで，$\bar{a}$ は a の "補元" すなわち，$a = 0$ なら $\bar{a} = 1$，$a = 1$ なら $\bar{a} = 0$ である．状態 q_1 では，M はその状態に貯えたものと異なる記号 (0 または 1) を飛ばして，右へ動き続ける．

(3) $a = 0, 1$ に対し，$\delta([q_1, a], B) = ([q_1, B], B, R)$．$M$ は最初の空白に到達すると，受理状態 $[q_1, B]$ に入る．

$\delta([q_0, a], a)$ $(a = 0, 1)$ に対する値は定義されない．したがって，M は，最初に有限制御部に貯えた記号に再び出会うと，受理状態に入ることなく停止する． ▨

8.3.2 多重トラック

もう一つの有用な技法は，テューリング機械のテープがいくつかのトラックから構成されるとみなすことである．各トラックに，一つずつの記号を保持することができ，TM のテープ記号は，トラックごとの記号の組になる．したがって，たとえば，図 8.13 のテープヘッドが読んでいるマスには，記号 $[X, Y, Z]$ がある．有限制御部での記憶の技法と同様に，多重トラックを利用してもテューリング機械の能力が拡張されるわけではなく，単に，テープ記号が有用な構造を持っているとみなすだけである．

例 8.7　多重トラックの一般的な使い方は，一つのトラックに入力を保持し，二つ目のトラックを“マーク入れ”として使うことである．各記号に“済み”の印を付けたり，データ中のいくつかの位置に印を付けることができる．例 8.2 と 8.4 はこの技法の二つの例であるが，テープがはっきりトラックから構成されているようには扱わなかった．今回の例では，非文脈自由言語

$$L_{wcw} = \{wcw \mid w \in (\mathbf{0} + \mathbf{1})^+\}$$

を認識するために，二つのトラックを陽に用いる．構成するテューリング機械は

$$M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ [q_1, B],\ [B, B],\ \{[q_0, B]\})$$

で，以下を満たす．

Q：状態の集合は $Q = \{q_1, q_2, \ldots, q_9\} \times \{0, 1, B\}$，すなわち，制御状態 q_i とデータ成分 0, 1, B の組である．ここでも，0,1 の入力記号を覚えておくために，有限制御部記憶の技法を使う．

Γ：テープ記号の集合は $\Gamma = \{B, *\} \times \{0, 1, c, B\}$ である．第 1 成分 (第 1 トラック) は，空白 B か印 $*$ のどちらかである．$*$ は中央の区切り記号 c の左右の文字列の照合のための検査済み印として用いる．テープ記号の第 2 成分は，本来のテープ記号である．すなわち，記号 $[B, X]$ $(X = 0, 1, c, B)$ をテープ記号 X と同じものとみなす．

Σ：入力記号は $[B, 0]$ と $[B, 1]$ である．これらは今述べたようにそれぞれ記号 0,1 と同一視される．

δ：遷移関数 δ は次の規則で定義される．以下 a と b は 0 または 1 を表す．

(1)　$\delta([q_1, B],\ [B, a]) = ([q_2, a],\ [*, a],\ R)$. 初期状態で，$M$ は読み取った記号 $a(= 0, 1)$ を有限制御部に貯え，それに済みの印を付け，右へ動く．テープ記号の第 1 成分を B から $*$ に書き換えることによって，そこに済み印を付ける．

(2)　$\delta([q_2, a],\ [B, b]) = ([q_2, a],\ [B, b],\ R)$. M は記号 c を探して右へ動く．a, b は 0 または 1 を表すが記号 c ではない．

(3)　$\delta([q, a],\ [B, c]) = ([q_3, a],\ [B, c],\ R)$. M は c を見つけると状態 q_3 に遷移し 1 つ右に動く．

(4)　$\delta([q_3, a],\ [*, b]) = ([q_3, a],\ [*, b],\ R)$. 状態 q_3 で M は印の付いたマスを読み飛ばす．

(5)　$\delta([q_3, a],\ [B, a]) = ([q_4, B],\ [*, a],\ L)$. 最初の印のない記号が有限制御部に貯えられた記号と同じなら，前半の 0, 1 ブロックの対応する記号と一致するので，その記号に済み印を付ける．M は状態 q_4 に遷移し，有限制御部に貯えた記号を削除し，左への移動を始める．

(6)　$\delta([q_4, B],\ [*, a]) = ([q_4, B],\ [*, a],\ L)$. M は済み印の付いた記号を読み飛ばして左へ移動する．

(7)　$\delta([q_4, B],\ [B, c]) = ([q_5, B],\ [*, a],\ L)$. M は記号 c に出会うと，状態 q_5 に切り替わりもう 1 マス左へ動く．状態 q_5 では，c のすぐ左の記号が検査済みか否かによって，次の動作を決める．済み印があれば，c の左にあるブロックの 0, 1 はすべて照合が終わったことになるので，c の右にある 0, 1 のすべてに済みの印があるかを調べ，検査済みででないものが残っていなければ受理状態に入る．c のすぐ左の記号が検査済みでなければ，最も左の未検査記号を見つけ出し，状態 q_1 から始まるサイクルを再開する．

(8)　$\delta([q_5, B],\ [B, a]) = ([q_6, B],\ [B, a],\ L)$. この規則は，$c$ の左の記号が検査済みでない場合を扱う．M は状態 q_6 に遷移し，済みの印を探して左へ動き続ける．

(9)　$\delta([q_6, B],\ [B, a]) = ([q_6, B],\ [B, a],\ L)$. 記号が未処理の間，$M$ は状態 q_6 に留まり左へ進む．

(10)　$\delta([q_6, B],\ [*, a]) = ([q_1, B],\ [*, a],\ R)$. 済みの印を見つけたら，

M は状態 q_1 に入り，最初の未検査記号を取り出すために1マス右に動く．

(11) $\delta([q_5, B],\ [*, a]) = ([q_7, B],\ [*, a],\ R)$. c から1マス左へ動いたときの状態 q_5 で検査済みの記号を読んだ場合を扱う．状態 q_7 に入り右へ1マス移動する．

(12) $\delta([q_7, B],\ [B, c]) = ([q_8, B],\ [B, c],\ R)$. 状態 q_7 で M は記号 c を見ている筈である．状態 q_8 に入りもう1マス右へ進む．

(13) $\delta([q_8, B],\ [*, a]) = ([q_8, B],\ [*, a],\ R)$. M は状態 q_8 のまま右へ動き続け検査済みの $0, 1$ を読み飛ばす．

(14) $\delta([q_8, B],\ [B, B]) = ([q_9, B],\ [B, B],\ B)$. 状態 q_8 のまま済み印のない $0, 1$ に出会うことなく空白 $[B, B]$ に到達すれば，M は入力を受理する．済み印のない $0, 1$ に出会った場合，c の前後のブロックは一致せず，M は受理状態に入らずに停止する． ▨

8.3.3 サブルーチン

一般のプログラム同様，テューリング機械を，相互作用する構成要素あるいは「サブルーチン」の集まりから組み立てられると便利である．テューリング機械のサブルーチンは，ある有用な処理を実行する状態の集合である．この状態集合には，開始状態と，とりあえずは動作を持たない別の状態とが含まれる．この状態は，他のサブルーチンに制御を渡すための "戻り状態" の働きをする．サブルーチンの "呼び出し" はその開始状態への遷移によって起こる．TM は "戻り番地"，すなわち，サブルーチンの終了後に遷移する状態を覚えておく機構を持たないので，設計した一つのサブルーチンをいくつかの状態から呼ぶためには，そのサブルーチンのコピーを，それぞれ異なる状態集合を使って作る必要がある．"呼び出し" はそれぞれ，コピーされた異なるサブルーチンの開始状態への遷移を引き起こし，各コピーはそれぞれ異なる状態に "戻る"．

例 8.8 "乗算" を実現する TM を設計しよう．設計する TM はテープ上に 0^n10^m をおいて計算を開始し，0^{mn} をテープ上に残して終了する．方針は：

(1) テープは一般に，$0^i10^n10^{kn}$ の形の列を持つ．

(2) 1回の基本サイクルでは，頭の0を B に変え，最後のグループに 0^n を追加し，$0^{i-1}10^n10^{(k+1)n}$ の形の列を得る．

(3) 結果として，毎回最初のグループの 0 を B に変えながら，0^n を後ろに m 回追加する．最初のグループの 0 をすべて B に変えた後では，最後のグループには，mn 個の 0 がある．

(4) 最後に，頭の 10^n1 を空白列に変えて，終了する．

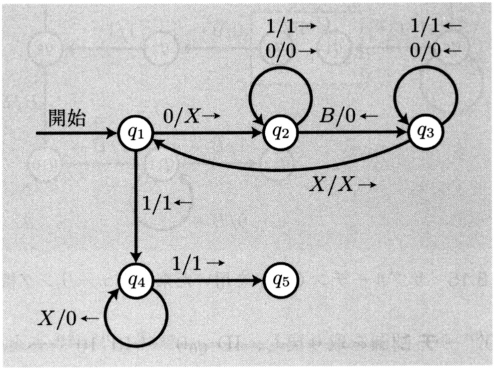


図 8.14 サブルーチン Copy

このアルゴリズムの中心は，Copy と呼ぶサブルーチンである．Copy は，0^n をテープの終端にコピーして，上記のステップ (2) を実現する．より正確には，Copy は ID を $0^{m-k}1q_10^n10^{(k-1)n}$ から $0^{m-k}1q_50^n10^{kn}$ に変える．図 8.14 はサブルーチン Copy の遷移図である．このサブルーチンでは，見ている 0 を X に変え，状態 q_2 のまま空白 B を見つけるまで右に動いて B を 0 に書き換え，次に状態 q_3 でマーカー X を見つけるまで左に戻る．このサイクルは状態 q_1 で 0 の代わりに 1 が見つかるまで繰り返され，1 が見つかった時点で，状態 q_4 を使って X をすべて 0 に戻し，状態 q_5 で終了する．

完全な "乗算" テューリング機械は状態 q_0 から出発する．TM が最初にすることは ID $q_00^m10^n$ から ID $0^{m-1}1q_10^n1$ に (何ステップかで) 遷移することである．必要な遷移は，図 8.15 中のサブルーチン呼び出しから左の部分に示されており，状態 q_0 と q_6 だけが使われる．

次に，図 8.15 のサブルーチン呼び出しの右側では，状態 q_7 から q_{12} が使われる．状態 q_7, q_8, q_9 の目的は，Copy が n 個の 0 をコピーした直後の ID

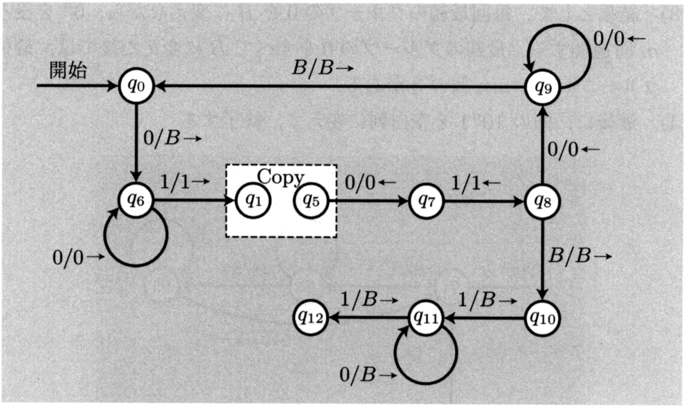


図 8.15　サブルーチン Copy を用いた乗算テューリング機械

$0^{m-k}1q_50^n10^{kn}$ から制御を取り戻し，ID $q_00^{m-k}10^n10^{kn}$ へと遷移することである．この時点で，サイクルが再開され，n 個の 0 をコピーするために再び Copy が呼ばれる．

例外として，状態 q_8 で TM は m 個の 0 をすべて空白に変えたこと (つまり $k=m$) を発見するかも知れない．その場合，状態 q_{10} への遷移が生じる．状態 q_{10} は状態 q_{11} の助けを借りて，先頭の 10^n1 を空白に変え，停止状態 q_{12} に入る．この時点で，TM の ID は $q_{12}0^{mn}$ になり，計算が終了する．▨

8.3.4　8.3 節の演習問題

♣ **!問 8.3.1**　問 8.2.2 で作った TM を 8.3 節で議論した技法を利用して作り直せ．

♣ **!問 8.3.2**　テューリング機械プログラムの共通演算の一つに“シフト”がある．理想的には，ヘッドの現在位置に新たなセルを追加してそこに文字を格納できるようにしたい．しかしテープにこのような編集をほどこすことはできないので，実際には，ヘッドの現在位置より右側にあるマスの中身をそれぞれ 1 マス右に移し，その後，現在位置へ戻って来なければならない．この演算を行う方法を示せ．
ヒント：ヘッドが戻ってくる位置を示す特別の記号を，現在位置に残せ．

♣ ***問 8.3.3**　TM のヘッドを現在位置から，1 または空白 B に出会うまですべての 0 を読み飛ばして右へ移動させるサブルーチンを設計せよ．ヘッドが現在読んでいる記号が 0 でなければ，TM は停止する．0,1,B 以外のテープ記号はないと仮

定してよい．次に，このサブルーチンを使って，二つ並んだ 1 を含まないような 0,1 の列を受理する TM を作れ．

8.4　基本テューリング機械の拡張

この節ではテューリング機械を拡張した計算機モデルで，これまで議論してきた TM の基本モデルと同等の計算能力を持つものを見てゆこう．中でも，多テープテューリング機械は重要である．なぜなら，多テープテューリング機械はこれまで学んできた 1-テープモデルに比べて，実際のコンピュータ (や他の種類のテューリング機械) をどのように模倣するかがずっとわかりやすいからである．しかし，追加されたテープは言語の受理能力に関する限りいかなる能力も付け加えることはない．

次に，非決定性テューリング機械を考える．これは与えられた状況 (状態とテープ記号の組) に対して，動作の有限個の選択が許されるように基本モデルを拡張したものである．この拡張もまた，テューリング機械による “プログラミング” を容易にするけれども，言語受理能力を増すことはない．

8.4.1　多テープテューリング機械

多テープテューリング機械を図 8.16 に示した．機械は有限制御部 (状態) と何本かのテープを持つ．各テープはそれぞれマスに分解され，各マスには有限テープアルファベット中の任意の記号が書ける．1-テープ TM と同様，テープ記号の集合は空白 B と入力記号を含み，空白は入力記号の集合には含まれない．状態集合には初期状態といくつかの受理状態が含まれる．最初，

(1)　入力，すなわち入力記号の有限列が 1 番目のテープ上に置かれる．

(2)　入力以外の，すべてのテープのすべてのマスは空白である．

(3)　有限制御部は初期状態にある．

(4)　1 番目のテープのヘッドは入力の左端に位置する．

(5)　他のテープヘッドは，すべて任意のマスに位置する．1 番目のテープ以外は，すべて完全に空白であるからすべてのマスは同じに見え，テープヘッドの初期位置は問題でない．

多テープ TM の動作は，状態とテープヘッドのそれぞれが読む記号によって定まる．1 回の動作で TM は以下のことを行う．

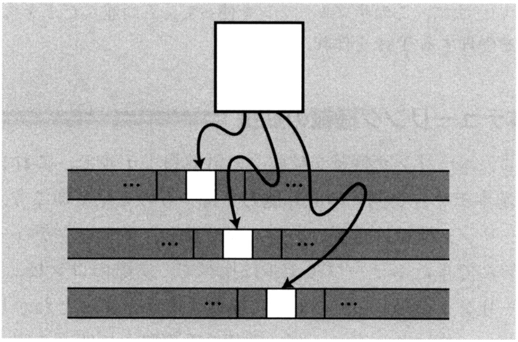


図 8.16　多テープテューリング機械

(1)　制御部は新しい状態に入る．その状態は直前の状態と同じであってもよい．

(2)　各テープ上で，新しいテープ記号が読まれているマスに書かれる．これらの記号は，以前そこにあった記号と同じであってもよい．

(3)　各テープヘッドは独立に，左，右，そのまま，の方向に動く．それゆえ，異なるヘッドは異なる方向に動いてもよいし，動かなくてもよい．

遷移規則の形式的記法については，ここでは述べないが，1-テープ TM の直接的な一般化として容易に理解できるだろう．ただし，ヘッドの移動方向を今度は L, R, S で表す．1-テープ機械の場合，ヘッドがその位置に留まることを許さなかったので，S という選択肢はなかった．多テープ TM の時点表示 (ID) の適当な記法を想像することも容易なので，形式的記法を定義することは省略する．多テープテューリング機械はは 1-テープ TM と同様，受理状態に入って入力を受理する．

8.4.2　1-テープテューリング機械と多テープテューリング機械の等価性

帰納的可算集合は 1-テープ TM で受理される集合として定義された．1-テープ TM は多テープ TM でもあるから，多テープ TM は帰納的可算集合を受理できる．しかし，帰納的可算でなく，しかも多テープ TM で受理される集合はあるだろうか．答は NO であり，そのことを，多テープ TM を 1-テープ TM

で模倣する方法を示すことによって証明する．

定理 8.9 多テープ TM で受理される言語は帰納的可算である．

証明 証明のアイデアを図 8.17 に示した．言語 L が k-テープ TMM で受理されたとする．M を $2k$-トラックを持つ 1-テープ TM N で模倣する．N のトラックのうち半分は M のテープ内容を保持し，残りの半分のトラックは対応するテープヘッドの現在位置を示す印を一つだけ持つ．図 8.17 は $k = 2$ の場合で，トラックの 2 と 4 はそれぞれ M のテープ 1 と 2 の内容を保持し，トラックの 1 と 3 は，それぞれテープ 1 と 2 のヘッド位置を保持する．

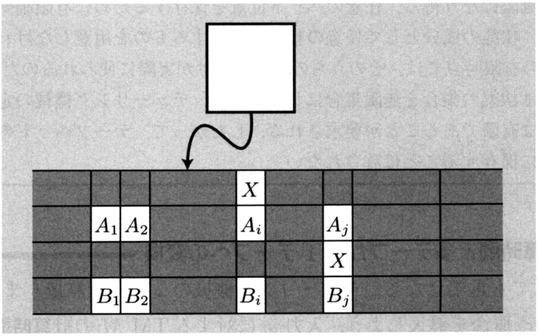


図 8.17 2-テープテューリング機械の 1-テープテューリング機械による模倣

M の動作を模倣するために，N のヘッドはすべての (k 個の) ヘッドマークを訪れなければならない．N は迷子にならないように自分の左に何個のヘッドマークがあるか常に記憶していなければならず，その個数は有限制御部の状態として保存される．N は M の状態を自分の有限制御部に保存している．さらに，各ヘッドマーカーを訪れた後，N は M の各ヘッドが読んでいる入力記号がわかり，M がどんな動作をするかがわかる．

そこで，N はそのテープ上の各ヘッドマーカーを再訪し，M の対応するテープを表すトラック上の記号を書き換え，必要ならヘッドマーカーを左右に移動させる．最後に，N は有限制御部に保存してある M の状態を変更する．これで M の 1 動作の模倣が終了する．

N の受理状態としては，保存している M の状態が M の受理状態であるようなものを選ぶ．こうして，入力は，模倣されている M が受理すれば N も受理し，さもなければ N も受理しない． ▨

有限性に関する注意

よくある誤解は，値が常に有限であることと，値が有限の集合に属する場合との混同である．今述べた多テープから 1-テープへの変換はこの違いの正しい理解に役立つ．変換の中で，テープヘッドの位置を記録するトラックを使った．これらの位置を整数として有限制御部に貯えることができないのはどうしてだろうか．不注意にも，n 回の動作の後では，TM のテープヘッドの位置は，開始位置から高々 n しか離れていないので，状態には n 以下の整数値を覚えておけばよいと考えてしまうかもしれない．

問題は次の点にある．位置の値は常に有限であるけれども，可能なヘッド位置の集合は無限になり得る．任意のヘッド位置を保存するために有限制御部を使うとしたら，状態の成分として任意の整数を値とするものを用意しなければならない．任意の有限時点では，そのうちの有限個だけが実際に使われるのだとしても，この成分は状態の集合を無限集合にしてしまう．テューリング機械の定義では状態の集合は有限であることが要求される．したがって，テープヘッドの位置を有限制御部に保存することは許されない．

8.4.3 計算時間と多テープから 1-テープへの変換

あとでとても重要になる，テューリング機械の「時間計算量」または「計算時間」という概念を導入しよう．入力 w に対する TM M の**計算時間** (running time) とは，M が停止するまでに要するステップ数である．M が w 上で停止しない場合は，w に対する M の計算時間は無限大であるとする．TM M の**時間計算量** (time complexity) は，関数 $T(n)$ でその値は長さ n のすべての入力 w に対する M の計算時間の最大値で定義される．すべての入力に対して停止するとは限らない TM の場合，$T(n)$ はいくつかのあるいはすべての n に対して無限大になる．しかし，特に注目するのは常に停止する TM，それも多項式の時間計算量 $T(n)$ を持つ TM であり，10.1 節で議論する．

定理 8.9 の構成は一見気が利かないように思える．実際，作られた 1-テープ TM はもとの多テープ TM よりずっと計算時間がかかる．しかし，二つのテューリング機械の計算時間は広い意味で同等であり，多テープ TM を模倣する 1-テープ TM の計算時間はもとの TM の計算時間の 2 乗を越えない．“2 乗” の上界は強い保証にはならないが，多項式の計算時間は保存される．10 章では次のことを示す．

(1) 計算時間における多項式時間の増大率とそれより大きな増大率とは，現実的な意味で，コンピュータで解けるものと解けないものとを分ける．

(2) 詳細な研究にも関わらず，必要な計算時間が多項式に近い時間にはなっていない問題が多数ある．したがって，問題を解くのに 1-テープ TM を使うか多テープ TM を使うかは，そのような問題を解くのに必要な計算時間を調べる際に本質的な問題とはならない．

1-テープ TM を多テープ TM の計算時間のお互いの 2 乗以内であることの議論は以下の通りである．

定理 8.10 定理 8.9 の 1-テープ TM N が k-テープ TM M の n 動作を模倣するのに必要な計算時間は $O(n^2)$ である．

証明 M の n 動作の後，テープヘッドマーカーが互いに $2n$ マスより遠く離れることはない．したがって，N が最も左にあるマーカーから出発したとすると，すべてのヘッドマーカーを訪問するのに $2n$ マスより多く右に動くことはない．次に N は左に動きながら，M を模倣するテープの内容を書き換えていき，必要ならマーカーの位置を左または右に移動させる．これに必要な計算時間は，$2n$ マスの左移動に加えて，(M のテープヘッドが右に動いた場合の) 逆方向への移動とマーカー X を書くための高々$2k$ 回の動作とである．

したがって，M の最初の n 動作の一つを模倣するのに必要な N の動作数は $4n+2k$ 以下である．この数は，k が模倣される動作数 n とは独立の定数であることから，$O(n)$ である．結局，N が M の n 動作を模倣する計算時間は，これの n 倍である $O(n^2)$ 以下である． ▨

8.4.4 非決定性チューリング機械

非決定性チューリング機械 (nondeterministic Turing machine, NTM) が今まで学習してきた決定性 TM と異なる点は，各状態 q と各テープ記号 X に対する遷移関数の値 $\delta(q, X)$ が，三つ組の有限集合

$$\{(q_1, Y_1, D_1),\ (q_2, Y_2, D_2),\ \ldots,\ (q_k, Y_k, D_k)\}$$

となる点だけである．NTM は，各ステップで，これらの三つ組のどれかを次の動作に選べる．しかし，状態 q_h をどれかから選び，テープ記号 Y_i を他から選び，方向 D_j をまた別の組から選ぶということはできない．

NTM M で受理される言語は，容易に想像できるように，すでに学習した NFA や PDA といった他の非決定性の機械と同じように定義される．すなわち，

M が入力 w を受理するのは，w を入力列とする初期 ID から受理状態を含む ID へ至るような動作の選択列が存在するときである．NFA や PDA の場合と同様に，受理状態に至らないような別の選択列があっても関係ない．

NTM は決定性 TM(DTM) で受理できない言語は受理できない．その証明では，NTM M_N に対し，任意の選択列によって M_N が到達可能なすべての ID を調べ上げる DTM M_D を構成できることを示す．M_D は，受理状態を含む ID を見つけることができれば，自分自身の受理状態に入る．M_D は，新しい ID をスタックでなくキューに置き，$k = 1, 2, \ldots$ に対し，ある有限時間内に M_N の k ステップまでを機械的にすべて模倣できるようにしなければならない．

定理 8.11 非決定性テューリング機械 M_N に対し，$L(M_N) = L(M_D)$ となる決定性テューリング機械 M_D が存在する．

証明 M_D は図 8.18 に描いたような多テープ TM として設計される．M_D の第 1-テープは M_N の ID の列を保持するキューである．M_N の ID の一つは "現"ID としてマークされ，それに続く ID が探索の処理対象となる．図 8.18 において，3 番目の ID3 が，ID の区切りに使われている星印 $*$ の上につけられた記号 X でマークされている．現 ID の左に並ぶ ID はすでに処理済であり，これ以降無視される．

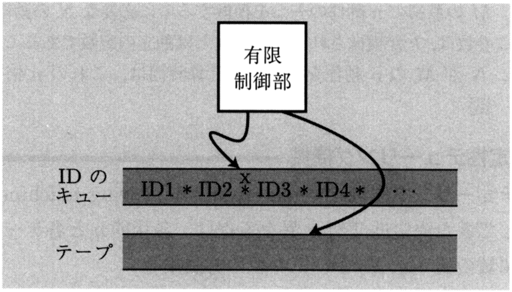


図 8.18 DTM による NTM の模倣

現 ID を処理するために，M_D は以下の動作を行う．

(1) M_D は現 ID の状態と読込記号を調べる．M_D の有限制御部には，状態と読込記号の組の各々に対して，M_N が持つ動作の選択肢の全体が組み込まれる．現 ID の状態が受理状態なら，M_D は入力を受理し M_N の模倣を停止する．

(2) 状態が受理状態でなく，状態と読込記号の組み合わせが k 個の動作を持っていれば，M_D は 2 番目のテープに現 ID のコピーを作り，それを使って，1 番目のテープの後ろに，現 ID の k 個のコピーを付け加える．

(3) M_D は新たに付け加えた k 個の ID を，M_N が持つ k 個の異なる動作のそれぞれにしたがって，変更する．

(4) M_D はマークされた現 ID に戻ってマークを消し，マークを右隣の ID に移す．次に (1) に戻り上記のステップを繰り返す．

M_D が受理状態に入るのは M_N が受理 ID に到達可能なことを見つけたときのみであるという意味で，模倣が正しいのは明らかだろう．しかし，確認しなければならないのは，M_N が n ステップの動作の後，受理 ID に到達するならば，M_D もいつかはその ID を現 ID とし，入力を受理するということである．

m を，各状況で M_N が持つ動作の選択肢の個数の最大値とする．このとき，M_N が持ち得る初期 ID の個数は 1 で，1 動作で到達できる ID は最大 m 個，2 動作で到達できる ID は最大 m^2 個，等々である．したがって，n 動作の後，M_N は高々 $1 + m + m^2 + \ldots + m^n$ 個の ID に到達できる．この数は nm^n 以下である．

M_D が M_N の ID を探索する順番は "幅優先" である．すなわち，まず 0 ステップで到達する ID (つまり初期 ID) を調べ，次に 1 ステップで到達する ID をすべて調べ，さらに 2 ステップで到達する ID を調べる，等々．特に，M_D は到達に n ステップより多くかかる ID を処理する前に，n ステップ以内で到達するすべての ID を現 ID として調べ，それに続く ID を作り出す．

結論として，M_N が n ステップ以内で受理 ID に到達するならば，その受理 ID は M_D によって処理される最初の nm^n 個の ID の中に含まれる．ここでは M_D が M_N の受理 ID を有限時間内に処理することだけが問題なので，この上界で十分である．したがって，M_N が入力を受理すれば M_D も受理する．M_D が入力を受理するのは M_N も受理するときだけであることはすでに確認したので，$L(M_N) = L(M_D)$ である． ▨

8.4.5 8.4 節の演習問題

♣ **問 8.4.1** 問 8.2.2 の言語を受理する多テープテューリング機械をそれぞれ非形式的にしかし明確に記述せよ．作成するテューリング機械は入力の長さに比例する時間で動くようにせよ．

♣ **問 8.4.2** 遷移関数が以下の表で与えられる非決定性 TM $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$ がある．
以下の入力上で到達可能な ID をすべて示せ．

δ	0	1	B
q_0	$\{(q_0,1,R)\}$	$\{(q_1,0,R)\}$	$\varnothing$
q_1	$\{(q_1,0,R),(q_0,0,L)\}$	$\{(q_1,1,R),(q_0,1,L)\}$	$\{(q_2,B,R)\}$
q_2	$\varnothing$	$\varnothing$	$\varnothing$

*a)　01

b)　001

♣ !**問 8.4.3**　次の言語を受理する非決定性(もし必要なら多テープ)テューリング機械を非形式的にしかし明確に記述せよ．非決定性を利用して繰り返しを避け，非決定性のもとでの計算時間を節約するようにせよ．すなわち，NTM の非決定性の分岐を多用し，分岐の深さは浅くなるようにせよ．

*a)　長さ 100 の繰り返し列を含む，0,1 の列から成る言語．繰り返し部分は連続していなくてもよい．形式的には，この言語は $wxyxz$ の形をした 0,1 の列から成り，$|x| = 100$ で w, y, z の長さは任意である．

b)　$w_1\#w_2\#\ldots\#w_n$ の形の列で，n は任意，各 w_i は 0,1 の列，ある j に対し w_j は j の 2 進表示であるもの全体から成る言語．

c)　(b) と同じ形の列で，少なくとも二つの j に対して w_j が j の 2 進表示に成っている列全体の言語．

♣ !**問 8.4.4**　非決定性テューリング機械

$$M = (\{q_0, q_1, q_2, q_f\},\ \{0,1\},\ \{0,1,B\},\ \delta,\ q_0,\ B,\ \{q_f\})$$

を考える．δ が次の規則から成るとき，$L(M)$ を非形式的にしかし明確に述べよ．$\delta(q_0,0) = \{(q_0,1,R),(q_1,1,R)\}$; $\delta(q_1,1) = \{(q_2,0,L)\}$; $\delta(q_2,1) = \{(q_0,1,R)\}$; $\delta(q_1,B) = \{(q_f,B,R)\}$.

♣ ***問 8.4.5**　テープが両方向無限の非決定性テューリング機械を考える．ある時点で，1 つのマスに記号 \$ があるのを除いてテープは完全に空白であるとする．現在，ヘッドはある空白マスにあり，状態は q であるとする．

a)　この NTM が \$ を見つけて状態 p に入ることを可能にする遷移規則を書け．

!b)　TM が決定性であるとする．\$ を見つけて状態 p に入るにはどうすればよいか．

♣　**問 8.4.6**　同数の 0 と 1 から成る列を受理する次のような 2-テープ TM を設計せよ．第 1 テープは入力を保持し，左から右へ読まれる．第 2 テープは，それまでに読んだ 0 と 1 の個数の差を貯えるのに使われる．状態，遷移規則，および各状態の直観的な役割を記述せよ．

♣　**問 8.4.7**　この問題では，特殊な 3 テープ TM を用いてスタックを実現する．

(1)　第 1 テープは入力を保持し読み込むために使われる．入力アルファベットは，スタックからのポップを意味する ↑ と，それぞれ記号 a, b をスタック

ヘプッシュすることを意味する a, b から成る．

(2)　第 2 テープはスタックを保持するのに用いる．

(3)　第 3 テープは出力テープである．記号がスタックからポップされるたびに，その記号が出力テープ上のすでに書かれた記号列の後ろに書き足される．

テューリング機械は空のスタックから出発し，入力に記述されたプッシュやポップの動作を，左から右に読みながら模倣する．入力が空スタックからのポップを指示する場合には，特別のエラー状態 q に入って停止する．入力全体が最後に空のスタックを残す場合には，最終状態 q_f に入って入力を受理する．この TM の遷移規則を非形式的にしかし明確に記述せよ．また，使った状態の目的を簡潔に説明せよ．

♣ **問 8.4.8**　図 8.17 において，1-テープ TM による k-テープ TM の一般的な模倣例を見た．

***a)**　この技法を 7 個のテープ記号を持つ 5-テープ TM を模倣するのに使うとき，1-テープ TM は何個のテープ記号を持つか．

***b)**　k 本のテープを 1 本で模倣する別のやり方は，最初の k トラックで k 本のテープを模倣し，第 $k+1$ トラックを k 個のヘッド位置を管理するのに使う方法である．第 $k+1$ トラックでは，テープヘッドを区別し，かつ 2 個以上のヘッドが同じマスにいることを許さなければならないことに注意せよ．この方法で，1-テープ TM に必要なテープ記号の数を減らすことができるか．

c)　k 本のテープを 1 本で模倣するまた別のやり方では，ヘッド位置をまとめて記憶しておくことをせず，第 $k+1$ トラックは一つのマスをマークするだけに使われる．どの時点においても，各トラックには，ヘッドがマークされたマスに揃うように，模倣されるテープが置かれる．k-テープ TM がヘッド i を動かすと，それを模倣する 1-テープ TM は i 番目のトラックの空白でない中身全体をヘッドの動きとは逆方にスライドさせ，マークされたマスが第 i テープヘッドによって読まれているマスを保持し続けるようにする．この方法は，1-テープ TM のテープ記号の個数を減らすのに役立つか．また，すでに議論した他の方法に比べて何か短所はあるか．

♣ **! 問 8.4.9**　k-ヘッドテューリング機械は 1 本のテープ上に k 個の読み書きヘッドを持つ．この TM の動作は，状態と各ヘッドが読んでいる記号によって定まる．1 動作で，TM は状態を変え，各ヘッドが見ているマスに新しい記号を書き，各ヘッドを右または左に動かすかまたはその位置に留めるかすることができる．いくつかのヘッドが同じマスを見ることもできるので，ヘッドは 1 から k まで番号付けられているものとし，与えられたマスに位置するヘッドのうち，一番番号の

大きいヘッドが書く記号が，実際にそのマスに残る記号であるとする．k-ヘッドテューリング機械で受理される言語は，通常の TM で受理される言語と同じであることを示せ．

♣!! **問 8.4.10 2 次元** (two-dimensional) テューリング機械は，通常の有限制御部と上下左右に無限に伸びた 2 次元格子のマスから成るテープを持つ．入力は格子の一つの行に置かれ，通常通り，ヘッドは最初一番左の非空白記号上に置かれ，状態は開始状態をとる．受理は最終状態に入ることによって通常通り定義される．2 次元テューリング機械で受理される言語は，通常の TM で受理されるものと同じであることを示せ．

8.5 制限されたテューリング機械

前節では，テューリング機械の外見上の，実際には言語認識能力を増加させない拡張を調べてきた．ここでは，TM の明白な制限で，同じく言語認識能力を減少させないものを調べる．最初の制限はささいなしかし後で出てくる多くの構成に役立つものであり，両方向無限のテープを，右方向にだけ無限に伸びるテープに代える．この制限された TM ではテープ記号を空白に置き換えることも禁止する．これらの制限の価値は，ID が非空白記号だけから成り常に入力の左端から始まると仮定できることにある．

次にある種の多テープテューリング機械で，プッシュダウンオートマトンの一般化に相当するものを調べる．まず，TM のテープをスタックとして振舞うように制限する．次に，テープを“カウンター”に制限する．すなわち，テープは 0 以上の整数を表すだけで，TM は 0 と正の数とを区別することだけができる．この議論の衝撃は，非常に単純なオートマトンでかつ任意のコンピュータとまったく同じ能力を持つものが存在するということである．第 9 章に見るテューリング機械の決定不能問題はこれらの単純な機械においても決定不能である．

8.5.1 半無限テープを持つテューリング機械

テューリング機械のテープヘッドはこれまでその初期位置から左右に動くことを許していたが，この節ではヘッド位置をその初期位置から右側の部分だけに制限する．実際，テープは**半無限** (semi-infinite)，すなわち初期ヘッド位置から左にはマスがないと仮定する．次の定理では，半無限テープを持つ TM が

通常の両方向無限テープを持つ TM を模倣できることを示す構成を与える．

その構成を支える技法は，半無限テープ上で二つのトラックを使うことである．上段はもとのテープの初期ヘッド位置とそこより右側の部分を表し，下段は初期位置より左の部分を逆順に表す．正確な配置を図 8.19 に示した．上段トラックはマス $X_0, X_1, \ldots$ を表す．ここで X_0 はヘッドの初期位置であり，$X_1, X_2, \ldots$ はその右に並ぶマスである．下段のマス $X_{-1}, X_{-2}, \ldots$ はヘッドの初期位置から左に並ぶマスを表す．下段トラックの左端にある $*$ 印は，左端記号の役割を果たし，半無限テープ TM のヘッドがテープの左端から飛び出すのを防ぐ．

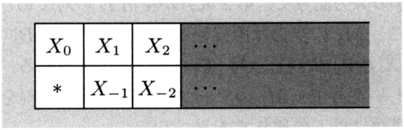


図 8.19　半無限テープで両無限テープを模倣する

TM にもう一つ，決して空白を書かないという制限を加えよう．この単純な制限を半無限テープに加えると，テープは常に，有限の非空白記号列の後ろに無限の空白が続くという形になる．さらに，非空白列は常にヘッドの初期位置 (テープの左端) から始まる．定理 9.19 や定理 10.9 では，ID がこの形をしていると仮定できることがいかに便利かわかるだろう．

定理 8.12　任意の TM M_2 で受理される言語は次のように制限された TM M_1 でも受理できる．

(1)　M_1 のヘッドはその初期位置の左側には移動しない．

(2)　M_1 は空白を決して書かない．

証明　条件 (2) はやさしい．空白のような働きはするが空白 B でない新しい記号 B' を作る．すなわち，

(a)　M_2 が規則 $\delta_2(q, X) = (p, B, D)$ を持てば，それを $\delta_2(q, X) = (p, B', D)$ に変える．

(b)　すべての状態 q に対して $\delta_2(q, B')$ を $\delta_2(q, B)$ と同じにする．

条件 (1) はもう少し手間がかかる．TM

$$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2.B, F_2)$$

は (2) の修正が行われており，空白記号を決して書かないものとする．

$$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B,B], F_1)$$

を次のように構成する．

Q_1：M_1 の状態集合は $\{q_0, q_1\} \cup (Q_2 \times \{U, L\})$ である．すなわち，M_1 の状態は初期状態 q_0，別の状態 q_1，そして M_2 の状態に第 2 成分として U(上段) または L (下段) を付けたものである．第 2 成分は，図 8.19 にあるように M_2 が読んでいるトラックが上段か下段かを示す．いい換えれば，U は M_2 のヘッドがその初期位置かまたはその右にあることを意味し，L は初期位置より左にあることを意味する．

Γ_1：M_1 のテープ記号は Γ_2 の記号の対で，$\Gamma_2 \times \Gamma_2$ である．M_1 の入力記号は，第 1 成分が M_2 の入力記号で，第 2 成分が空白 B の対であり，$[a, B]$ $(a \in \Sigma)$ の形をしている．M_1 の空白は両成分とも空白の対 $[B, B]$ である．さらに，すべての記号 $X \in \Gamma_2$ に対し，対 $[X, *]$ が Γ_1 に含まれる．ここで，$*$ は Γ_2 に属さない新しい記号で，M_1 のテープの左端記号の役割を果たす．

δ_1：M_1 の遷移関数は以下の通りである．

(1)　任意の $a \in \Sigma$ に対し $\delta_1(q_0,\ [a, B]) = (q_1,\ [a, *],\ R)$．$M_1$ の最初の動作は $*$ 印を一番左のマスの下段に置く．状態を q_1 にし，ヘッドは，左への移動や留まることは許されないので，右へ動く．

(2)　任意の $X \in \Gamma_2$ に対し $\delta_1(q_1, [X, B]) = ([q_2, U],\ [X, B],\ L)$．状態 q_1 で M_1 は M_2 の初期 ID に移る．ヘッドを左に戻しで状態 $[q_2, U]$ に遷移し，M_2 の初期状態で M_1 の上段トラックを読む．

(3)　$\delta_2(q, X) = (p, Y, D)$ ならば，すべての $Z \in \Gamma_2$ に対して

(a)　$\delta_1([q, U],\ [X, Z]) = ([p, U],\ [Y, Z],\ D)$ かつ

(b)　$\delta_1([q, L],\ [Z, X]) = ([p, L],\ [Z, Y],\ \overline{D})$,

ただし $\overline{D}$ は D の逆方向を表し，$D = R$ ならば $\overline{D} = L$，$D = L$ ならば $\overline{D} = R$ である．M_1 は，左端のマスにいるのでなければ，M_2 を U, L で示されたトラック上で模倣する．しかし，M_2 のテープの左半分は折り返されて M_1 のテープの下段トラックに逆順に収められているから，下段トラックでは，M_1 は M_2 と逆方向に動くことに注意せよ．

(4)　$\delta_2(q, X) = (p, Y, R)$ ならば，$\delta_1([q, L],\ [X, *]) = \delta_1([q, U],\ [X, *]) = ([p, U],\ [Y, *],\ R)$．この規則は左端記号 $*$ の扱い方の一例を示している．M_2 がヘッドの初期位置から右へ動くとき，それ以前にその左にいたか右にいたかに関わらず，つまり，第 2 成分が L であるか U であるかに関わらず，M_1 は右へ動き，上段トラックを読む．すなわち，M_1 は図 8.19 の X_1 で表される位置に動く．

(5)　$\delta_2(q, X) = (p, Y, L)$ ならば，$\delta_1([q, L],\ [X, *]) = \delta_1([q, U],\ [X, *]) =$

$([p, L], [Y, *], R)$．この規則は上の規則と同様であるが，M_2 がその初期ヘッド位置から左に動く場合を扱っている．M_1 は左端から，右に動かなければならず，今度は下段トラックに移り，図 8.19 の X_{-1} のマスを読む．

F_1：受理状態の集合 F_1 は $F_2 \times \{U, L\}$ である．すなわち，M_1 の状態で，第 1 成分が M_2 の受理状態であるもの全体である．M_1 は受理するときに上段を見ていても下段を見ていても構わない．

定理の証明は本質的には終わっている．M_2 の動作数に関する帰納法で，下段トラックを逆順に置きその後ろに上段トラックをつなげたもので M_1 が M_2 の ID を模倣できることを示せる．さらに，M_1 が受理状態に入るのはまさに M_2 が受理状態に入るとき，そのときだけである．したがって，$L(M_1) = L(M_2)$ が成り立つ．▨

8.5.2 多スタック機械

プッシュダウン・オートマトンを一般化し，複数のスタックを許すと何が起こるか，調べよう．例 8.7 より，テューリング機械は，どんな 1 本スタックの PDA でも受理されない言語を受理できることがすでにわかっているが，本節では，PDA に 2 本のスタックを付けると TM が受理する任意の言語を受理できることを示す．

次の節では,「計数機械」と呼ばれる機械のクラスを調べる．計数機械は，有限個数の整数 (カウンタと呼ばれる) を記憶する能力を有するだけであり，どの整数が現在 0 であるかということだけで，動作を選択する．計数機械はカウンタに 1 を加えたり引いたりすることができるだけであり，二つの 0 でないカウンタの値が異なるか否かを直接答えることはできない．実際，カウンタは二種類の記号だけを置くことのできるスタックとみなせる．一つは底記号で，スタックの一番底に置かれるだけであり，もう一つがスタックにプッシュ，ポップされる．

多スタック機械の形式的な取り扱いは述べないが，アイデアを図 8.20 に示した．k-スタック機械は k 本のスタックを持つ決定性 PDA である．入力は，TM のようにテープ (スタック) 上に置かれるのではなく，PDA のように入力源から受け取る．多スタック機械は有限制御部で有限個の状態の一つをとり，有限のスタック・アルファベットを持ち，すべてのスタックで共通に使う．さらに，多スタック機械の動作は，以下に情報に基づいて定まる．

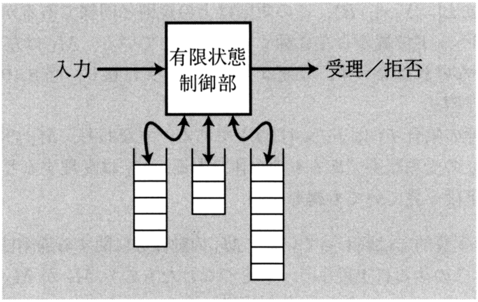


図 8.20　3 本のスタックを持つ機械

(1)　有限制御部の状態．

(2)　有限の入力アルファベットから選ばれた入力記号．代わりに，ε 入力を使った動作を行うこともできるが，機械を決定性にするために，どの状態においても ε 動作と非 ε 動作が同時に定義されることはない．

(3)　各スタックのトップ記号．

1 回の動作で，多スタック機械は以下のことを行う．

(a)　新しい状態に変わる．

(b)　各スタックのトップ記号を 0 個以上のスタック記号の列で置き換える．各スタックで異なる置き換えが可能であり，それが普通である．

したがって，k-スタック機械の遷移規則は，

$$\delta(q, a, X_1, X_2, \ldots, X_k) = (p, \gamma_1, \gamma_2, \ldots, \gamma_k)$$

という形をしている．この規則によれば，状態 q で $i = 1, 2, \ldots, k$ に対し X_i が第 i スタックのトップにあるとき，機械は入力 a (入力記号または ε) を消費して状態 p に移り，$i = 1, 2, \ldots, k$ に対し第 i スタックのトップの X_i を列 γ_i に置き換える．多スタック機械は受理状態に入ると入力を受理する．

決定性機械による入力処理を簡単にするために，一つの機能を付け加えよう．終端印と呼ばれる特別の記号 \$ があって，入力の最後にだけ現れ，入力列の一部ではないものとする．終端印により，いつすべての可能な入力を読み終わったかがわかる．次の定理で，多スタック機械がテューリング機械を模倣するのに，終端印がいかに役立つかがわかる．TM では，最初の空白記号が入力の終

わりを示すので，終端印を必要としない．

定理 8.13　言語 L がテューリング機械で受理されるならば，L は 2-スタック機械で受理される．

証明　本質的なアイデアは，2 本のスタックで TM の 1 本のテープを模倣できる点にある．左右両端の非空白記号より外側の無限の空白部分を除き，一方にヘッドの左側の部分を入れ，もう一方に右側の部分を入れる．より詳しくは，L を受理する (1-テープ) TM M を 2-スタック機械 S は以下のように模倣する．

(1)　S は 2 本のスタックに**スタック底印** (bottom-of-stack marker) を置いた状態から始める．この目印は，スタックの開始記号として使われるが，スタックの他の場所には現れない．以下では，スタックが底印だけを含むとき,「スタックは空である」と表現する．

(2)　$w\$$ が S の入力だとする．S は，入力の終端印 \$ を見るまで，w を第 1 スタックにコピーする．

(3)　S は第 1 スタックから順に記号をポップし，第 2 スタックにプッシュする．この時点で，第 1 スタックは空になり，第 2 スタックには w が，左端を頭にして入っている．

(4)　S は (模倣される)M の初期状態に遷移する．第 1 スタックは空で，このことは，テープヘッドの左側のマスには空白以外なにもないことを示している．S は第 2 スタックに w を保持し，このことは，w がテープヘッドの右側のマスにあることを示している．

(5)　S は M の動作を次のように模倣する．

　(a)　S は，M の状態をその有限制御部に貯えているので，M の状態を知っており，それを q とする．

　(b)　S は M のテープヘッドが読んでいる記号 X を知っている．それは S の第 2 スタックのトップである．第 2 スタックが空のときは例外で，M はまさに空白に到達したところであり，空白を見ているものと解釈する．

　(c)　したがって，S には M の次の動作がわかる．

　(d)　M の次状態は，S の有限制御部の成分として以前の状態の代わりに記録される．

　(e)　M がテープ上の X を Y で書き換えて右へ動くとき，Y はヘッドの左側にあるので，S は，Y を第 1 スタックにプッシュし X を第 2 スタックからポップする．しかし，2 種類の例外がある．

　　i.　第 2 スタックが空 (したがって X は空白記号) のとき，第 2 スタックは変化しない．これは M がさらに右へ動いて空白を読むこ

とに対応する．

ii. Y が空白記号で第 1 スタックが空のとき，M のヘッドの左側は依然として空白のままなので，第 1 スタックは空のままである．

(f) M がテープ上の X を Y で置き換え左へ動くとき，S は第 1 スタックのトップ記号 (Z とする) をポップし，第 2 スタックの X を ZY で置き換える．この操作は，ヘッドの一つ左にあったマスが，ヘッドの右にくることに対応している．例外として，Z が底印のとき，M は BY を第 2 スタックにプッシュし，第 1 スタックはポップしない．

(6) S は新しい状態が受理状態であれば入力を受理し，さもなければ M の次の動作を同じように模倣する． ▨

8.5.3 計 数 機 械

計数機械 (counter machine) には二つの捉え方がある．

(1) 計数機械は多スタック機械 (図 8.20) と同じ構造を持つが，スタックの代わりにカウンタがある．カウンタは任意の非負整数を保持するが，0 と正の数の区別がつくだけである．すなわち，計数機械の動作はその状態と入力記号，どのカウンタが 0 か，によって定まる．1 動作で計数機械は，

(a) 状態を変える．

(b) 各カウンタを独立に 1 だけ足したり引いたりできる．しかし，カウンタが負になることは許されず，0 のカウンタから 1 引くことは許されない．

(2) 計数機械は制限された多スタック機械とみなすことができる．制限は次のようなものである．

(a) スタック記号は二つだけであり，それを Z_0 (**スタック底印**) と X とする．

(b) Z_0 は最初に各スタックに置かれる．

(c) Z_0 は X^iZ_0 $(i \geq 0)$ の形の列とだけ置き換えてよい．

(d) X は X^i $(i \geq 0)$ の形の列とだけ置き換えてよい．すなわち，Z_0 はスタックの底にだけ現れ，他のスタック記号はあるとしても X だけである．

計数機械の定義には (1) を用いるが，スタック X^iZ_0 とカウンタ i とが同一視できることから，二つの定義は明らかに同じ機能の機械を定義する．実際，定

義 (2) では，スタックのトップ記号が，$i=0$ なら Z_0，$i>0$ なら X，となるので，0 と他の数とを区別できる．しかし，二つの正の数を区別することは，ともにトップ記号が X になることから，できない．

8.5.4 計数機械の能力

計数機械で受理される言語について，簡単だが有用な事実をいくつか見ておこう．

- 計数機械で受理される言語は帰納的可算である．実際，計数機械は多スタック機械の特殊な場合であり，多スタック機械は，多テープテューリング機械の特殊な場合であり，多テープテューリング機械が受理する言語は定理 8.9 より，帰納的可算言語に限られる．
- 1-計数機械が受理する言語は CFL である．カウンタは，(2) の見方によれば，スタックであり，1-計数機械は 1-スタック機械，すなわち，PDA の特殊な場合である．実は，1-計数機械で受理される言語は，決定性 PDA で受理されるが，その証明は驚くほど複雑である．難しさの原因は，多スタック機械や計数機械が入力の終端印 \$ を持つことである．非決定性 PDA は，入力を見終わり \$ を見ていることを推測することができる．したがって，終端印を持たない非決定性 PDA が終端印を持つ DPDA を模倣できることは明らかである．しかし，証明が難しくここで扱わないのは，終端印を持たない DPDA で終端印を持つ DPDA を模倣できることを示すことである．

計数機械に関する驚くべき結果は，テューリング機械を模倣しすべての帰納的可算言語を受理するのに，2-カウンタで十分なことである．これからこの結果を示すが，最初 3-カウンタで十分なことを示し，次に 3-カウンタ機械を 2-カウンタ機械で模倣する．

定理 8.14 すべての帰納的可算言語は 3-計数機械で受理される．

証明 すべての帰納的可算言語は 2-スタック機械で受理されることを述べた定理 8.13 から始めよう．次にスタックをどうやってカウンタで模倣するかを示す必要がある．使われるスタック記号が $r-1$ 個あるとする．これらの記号を 1 から $r-1$ までの数と同一視し，スタック $X_1X_2\ldots X_n$ を r 進数とみなす．すなわち，このスタックを (通常通り，左をスタックの頭と考えて)，整数 $X_nr^{n-1}+X_{n-1}r^{n-2}+\ldots+X_2r+X_1$ で表す．

二つのカウンタで，それぞれ2本のスタックを表し，3番目のカウンタを使って他の二つのカウンタを調整する．特に，カウンタ値に r を掛けたり割ったりするのに第3カウンタを利用する．

スタックに対する操作は3種類に分かれる．トップ記号のポップ，トップ記号の変更，スタックへの記号のプッシュ，である．2-スタック機械の1動作には，これらの操作がいくつか含まれる．たとえば，スタックのトップ記号 X を記号列に置き換える動作は，X の変更とそれに続く記号のスタックへのプッシュの列に分解される．数 i で表されたスタックに対するこれら3種類の操作は，以下のように遂行される．各操作に必要な r までの数を数えることは，多スタック機械の有限制御部を使ってできることに注意せよ．

(1) スタックのポップに対して，i を i/r に置き換え，余り X_1 を有限制御部に貯える．第3カウンタを0にして開始し，i から1ずつ引いていって r 回1を引くたびに第3カウンタを1増やすことを繰り返し，i を持っていたカウンタが0になったら，止める．このとき回数を数える有限制御部には余り X_1 が記録されている．次に，もとのカウンタを1増やして第3カウンタを1減らすことを，第3カウンタが再び0になるまで繰り返す．この時点で以前 i だったカウンタの値は i/r になる．

(2) 数 i で表されているスタックのトップ記号を X から Y に書き換えるには，i を少しだけ，実際 r 以下，増減すればよい．数として $Y > X$ ならば i を $Y - X$ 増やし，$X < Y$ ならば $X - Y$ 減らせばよい．

(3) i を保持しているスタックに X をプッシュするには，i を $ir + X$ にしなければならない．そのために，数 i を1減らすたびに第3カウンタ (常に最初は0) を r 増やすことを繰り返す．もとのカウンタが0になるとき，第3カウンタには ir が入っている．第3カウンタの内容を，(1) でやったようにもとのカウンタにコピーし，第3カウンタを0に戻す．最後に，もとのカウンタを X だけ増やす．

構成を完成させるためには，2-スタック機械の初期状態を表すために，カウンタの初期設定をしなければならない．このステップはスタックの開始記号に相当する1から $r-1$ までのどれかの数の分だけ，それぞれ対応するカウンタを増やせばよい．▨

定理 8.15 すべての帰納的可算言語は，2-計数機械で受理される．

証明 前定理により，3-カウンタを2-カウンタで模倣する方法を示せばよい．アイデアは三つの数 i, j, k を一つの数 $m = 2^i 3^j 5^k$ で表すことである．第1カウンタはこの数を保持し，第2カウンタは，三つの素数 2,3,5 のどれかによる乗除算の補助に用いる．3-計数機械の模倣には次の操作を行う．

カウンタ数を 3 から 2 に減らす際の定数の選択

定理 8.15 の証明における定数 2, 3, 5 が各々異なる素数であることがいかに重要かに注意しよう．たとえば $m = 2^i 3^j 4^k$ を選んだとしたら，$m = 12$ は $i=0$, $j=1$, $k=1$ を表すかも知れないし，$i=2$, $j=1$, $k=0$ を表すかも知れない．したがって，i や k が 0 か否かを判定することができず，3 カウンタ機械を模倣できなくなってしまう．

(1) i, j, k のどれかを 1 増やす．i を 1 増やすには，m を 2 倍すればよい．すでに定理 8.14 の証明で，第 2 カウンタを使って定数 r 倍する方法を示した．同様に，j を 1 増やすには m を 3 倍し，k を 1 増やすには m を 5 倍する．

(2) i, j, k がそれぞれ 0 か否か調べる．$i = 0$ かを調べるには，m が 2 で割り切れるか否かを決定しなければならない．m を第 2 カウンタにコピーしながら，計数機械の状態を使って，m からの 1 の減算回数が偶数回か奇数回かを調べる．$m = 0$ になるまでの減算回数が奇数回ならば，$i = 0$ である．判定 (コピー) が終わったら，第 2 カウンタの内容を第 1 カウンタに戻す．同様に，$j = 0$ か否かは，m が 3 で割り切れるか否かで判定し，$k = 0$ か否かは，m が 5 で割り切れるか否かで判定する．

(3) i, j, k のどれかを 1 減らす．そのためには m をそれぞれ 2,3,5 で割る．定理 8.14 の証明では，補助カウンタを用いて定数での割り算を行う方法を示した．3-計数機械は負の数を扱えないので，それを模倣する 2-計数機械は，割り算に余りが出る場合には，受理せずに停止する．

8.5.5 8.5 節の演習問題

♣ **問 8.5.1** 次の言語を受理する計数機械を非形式的にしかし明確に記述せよ．それぞれ，二つ以下のできるだけ少ないカウンタ数で実現せよ．

- ***a)** $\{0^n 1^m \mid n \geq m \geq 1\}$
- b) $\{0^n 1^m \mid m \geq n \geq 1\}$
- ***!c)** $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$
- **!!d)** $\{a^i b^j c^k \mid i = j \text{ or } i = k \text{ or } j = k\}$

♣!! **問 8.5.2** この問の目的は，入力に終端印が付いた 1-スタック機械が決定性 PDA (DPDA) 以上の能力を持たないことを示すことにある．$L\$$ は言語 L とただ一つの語 \$ から成る言語との連接 $L\$ = \{w\$ \mid w \in L\}$ を表す．$L\$$ が DPDA で受理されるならば，L も DPDA で受理されることを示せ．**ヒント**：この問題は，実際には，DPDA が問 4.2.2 で定義した演算 L/a で閉じていることを示す問題である．$L\$$ を受理する DPDA P のスタック記号 X を，状態の集合 S とのすべて

の可能な対 (X, S) で置き換え，P がスタック $X_1X_2 \ldots X_n$ を持てば，構成されたDPDA は $(X_1, S_1)(X_2, S_2) \ldots (X_n, S_n)$ を持つようにせよ．ただし S_i は，P が ID $(q, a, X_iX_{i+1} \ldots X_n)$ から受理に至るような状態 q の集合である．

8.6 テューリング機械とコンピュータ

テューリング機械と毎日使っている通常のコンピュータとを比較しよう．二つのモデルはかなり違って見えるけれども，まったく同じ帰納的可算言語を受理することができる．“通常のコンピュータ”という概念は数学的にきちんと定義されていないので，この節での議論は非形式的にならざるを得ない．特に，扱われる数が，コンピュータの設計に組み込まれた標準的な限界 (たとえば32ビット番地空間) を越えたときにコンピュータができることに関しては，直観に訴えざるを得ない．

この節の主張は二つに分かれる．

(1) コンピュータはテューリング機械を模倣できる．

(2) テューリング機械はコンピュータを模倣できる．それも，コンピュータのステップ数の多項式で抑えられる時間内に模倣できる．

8.6.1 コンピュータによるテューリング機械の模倣

最初に，コンピュータでテューリング機械を模倣する方法を示そう．与えられたTM M に対し，M のように振舞うプログラムを書かなければならない．M の一つの特徴は有限制御部である．有限個の状態と有限個の遷移規則しかないので，プログラムは状態を文字列で符号化し，それを見て動作を決定する遷移表を使うことができる．同様に，テープ記号も有限個しかないので，固定した長さの文字列で符号化できる．

重要な問題が生じるのは，テューリング機械のテープを模倣するときである．テープは無限に長く成長できるけれども，コンピュータの記憶装置，主記憶，ディスクその他の記憶装置，はすべて有限である．無限のテープを有限の量の記憶装置で模倣できるだろうか．

記憶装置を置き換えられる見込みがなければ，事実不可能である．その場合，コンピュータは有限オートマトンであり，受理できる集合は正則集合に限られる．しかし，普通のコンピュータは，交換可能な記憶装置 (たとえばフロッピー

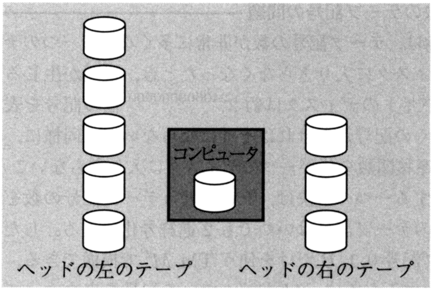


図 8.21　コンピュータによるテューリング機械の模倣

ディスク) を備えている．実際，典型的なハードディスクは取り外し可能で，空のディスクと交換可能である．

使用可能なディスクの数には，自明な上限はないから，コンピュータが必要とする限りいくらでも多くのディスクを使えると仮定しよう．こうして，ディスクを図 8.21 に示したように，二つのスタックに配置することができる．一つのスタックには，テューリング機械テープのテープヘッドの左に位置するマスの中のデータを保持し，もう一つのスタックはテープヘッドの右にあるデータを保持する．スタックを下がれば下がるほど，そこに保持されるデータはテープヘッドから遠ざかる．

TM のテープヘッドが，左に十分遠く動き，現在コンピュータに載せられているディスクにはないマスに到達したとき，コンピュータは "左と交換" というメッセージを印刷する．現在，載せられているディスクは，人間のオペレータによって取り外され，右スタックのトップに置かれる．さらに，左スタックのトップのディスクがコンピュータに載せられ，計算が再開される．

同様に，TM のテープヘッドが，ずっと右に動いて，載せられているディスクにないマスまできたときには，"右と交換" メッセージが印刷される．人間のオペレータは現在載っているディスクを左スタックのトップに置き，右スタックのトップのディスクをコンピュータに載せる．コンピュータが載せることを要求したディスクのスタックが空だったとき，コンピュータはテープの空白領域に入ったことを意味し，人間のオペレータは，お店に行って新しいディスクを買ってこなければならない．

非常に多数のテープ記号の問題

8.6.1 項の議論は，テープ記号の数が非常に多くなって一つのテープ記号を表す符号が一つのディスクに入りきらなくなったとき，疑問が生じる．実際には，たとえば 30 ギガバイトのディスクは約 $2^{240000000000}$ 個の記号を表すことができるので，非常に多くの記号がなければそうはならないが．同様に，状態数が非常に大きくなって，遷移関数全体が一つのディスクに入りきらないことも起こり得る．

この問題に対する一つの解決は，TM が使うテープ記号の数を制限することから始まる．任意のテープ記号はいつでも 2 進符号化できる．したがって，任意の TM M はテープ記号 $0, 1, B$ だけを使う TM M' で模倣できる．しかし，M' は，M の動作を模倣するために，M が読んでいる記号を表すすべてのビットを読んで有限制御部に貯えなければならないから，M よりずっと多くの状態を必要とする．こうして，非常の多くの状態が必要で，M' を模倣するコンピュータは，M' の状態が何で，次の動作が何であるかを決定するために何度もディスクを交換しなければならないかもしれない．この種の仕事を実行するコンピュータについて考えた人はいないので，典型的なオペレーティングシステムはこの種のプログラムをサポートしていない．しかし，望むなら，裸のコンピュータを直接プログラムしてこのような能力を備えさせることができる．

幸いにも，多数の状態やテープ記号を持つ TM をどのように模倣するかという問題は，もっと巧みに処理できる．9.2.3 項では，事実上「プログラム記憶方式」の TM を設計できることを示す．この TM は,「万能 TM」と呼ばれ，任意の TM に対して，その遷移関数をテープ上に 2 進符号化したものを受け取り，その TM を模倣する．万能 TM は，適当な数の状態とテープ記号を持つ．したがって，万能 TM を模倣することによって，ディスクに貯えられる上限を圧迫するような状態数の TM を模倣することなく，普通のコンピュータは望みの帰納的可算言語を受理できるようにプログラムできる．

8.6.2 テューリング機械によるコンピュータの模倣

逆方向の比較も考える必要がある．普通のコンピュータにできて，テューリング機械にできないことはあるだろうか．副次的ではあるがより重要な問題は，コンピュータはある種のことをテューリング機械よりずっと速く計算できるか，という問題である．この節では，テューリング機械がコンピュータを模倣できることを議論する．そして 8.6.3 項では，模倣が十分速く行えること，つまり，

与えられた問題に対して，コンピュータとテューリング機械と間には計算時間に多項式の差しか生じないことを示す．ここで，互いに多項式程度しか違わない計算時間をすべて同等と考えることには重要な意味があることを思い出そう．一方，指数的な計算時間の差は，“多すぎる”．第 10 章では，多項式対指数の計算時間の理論を取り上げる．

TM がどうやってコンピュータを模倣するかを調べる手始めに，典型的なコンピュータ操作の，現実的な，しかし，非形式的なモデルを与えよう．

(a) 最初に，コンピュータの記憶装置上の**語** (word) はいくらでも長い列から成り，語は**番地** (address) を持つと仮定する．現実のコンピュータでは，語は 32 または 64 ビットの長さであるが，ここでは与えられた語の長さに制限はつけない．番地は，整数 0,1,2,... であるとする．現実のコンピュータでは，個々のバイトが連続した整数で番号付けられ，語は 4 または 8 の倍数で番地付けされるが，その違いはあまり重要でない．さらに，実際のコンピュータでは「メモリー」中の語の数には上限があるが，任意個数のディスクや他の記憶装置中のデータも勘定に入れて，語の個数には制限がないものとする．

(b) コンピュータのプログラムはメモリー中の語に貯えられていると仮定する．これらの語は，それぞれ機械語やアセンブリ言語にあるような単純な命令を表す．たとえば，ある語から別の語にデータを移す，あるいはある語を別の語に加える，などである.「間接番地」は許される．それゆえ，命令は，参照している語の内容を，命令を適用する語の番地として使うことができる．この能力は，すべての近代的なコンピュータに見られ，配列処理を実行することやリストのリンクをたどること，さらに一般のポインタ操作などに必要である．

(c) 命令は，定まった有限個数の語を含み，高々一つの語の値を書き換える．

(d) 典型的なコンピュータは特別に高速なメモリーである**レジスター** (register) を持ち，可算のような演算は，しばしばレジスターだけに制限される．しかし，ここでは，そのような制限を設けず，任意の演算が任意の語に適用可能であるとする．異なる語に対する演算速度の差は勘定に入れないし，テューリング機械とコンピュータとの言語認識能力を比較するときには必要でさえない．計算時間を多項式の範囲内で比較するときにも，そ

の差は定数倍に“すぎない”ので，異なる語に対するアクセス速度の比は重要でない．

図 8.22 に，コンピュータを模倣するテューリング機械がどのように設計されるかを示した．この TM はいくつかのテープを持つが，8.4.1 項の技法を使って 1-テープ TM に変換できる．最初のテープは，コンピュータのメモリー全体を，メモリー語の番地に続いてその中身を符号化して，表す．番地と中身はともに 2 進で書かれる．* 記号と # 記号は目印で，それぞれ番地と中身の終わりを見つけ，2 進列が表すものが番地か中身かの判定を容易にするために使われる．別の印 $ は 番地とその中身を表す列の始まりを示す．

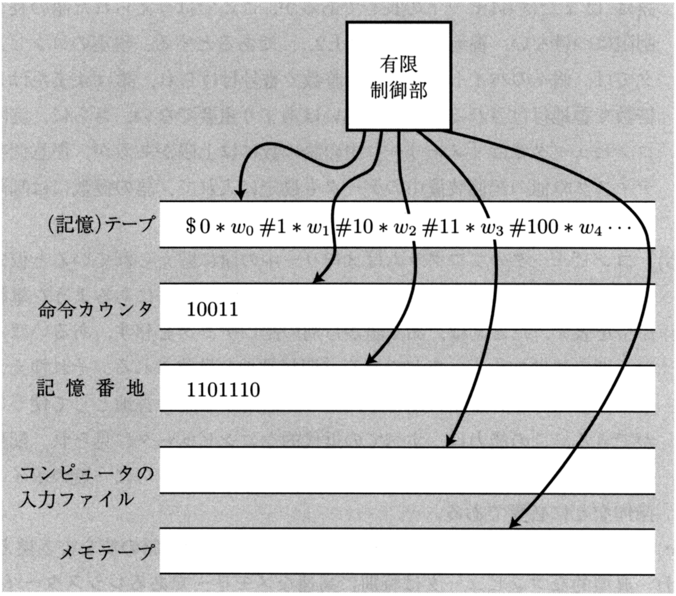


図 8.22　コンピュータを模倣するテューリング機械

第 2 テープは,「命令カウンタ」である．このテープは一つの整数を 2 進で保持し，テープ 1 の番地の一つを表す．この位置に貯えられている値は，次に実行されるコンピュータ命令として解釈される．

第3テープは,「メモリー番地」か，または，その番地がテープ1で見つかったあとでは，その番地の中身を表す．命令を実行するために，TM は計算に含まれるデータを保持する1個以上のメモリー番地の中身 (データ) を見つけなければならない．まず，欲しいデータの番地がテープ3にコピーされ，テープ1の番地と一致するまで比較される．この番地の中身は第3テープにコピーされ，必要とされる場所に移される．典型的には，コンピュータのレジスターに対応する，低い番地のメモリーに移される．

構成される TM はコンピュータの**命令サイクル** (instruction cycle) 以下の手順で模倣する．

(1) テープ1でテープ2の命令番号に一致する番地を探す．テープ1の左端の \$ 記号から出発して右に動き，各番地とテープ2の中身とを比べる．二つのテープの番地の比較はやさしい．二つのヘッドを右に並走させ，見ている記号がずっと一致するかを調べればよい．

(2) 命令番地が見つかったら，その番地にある語を調べる．語が命令を表すときには，最初の数ビットは，(コピー，加算，枝分かれ等の) 動作を表し，残りのビットはその動作に必要な番地である．

(3) 命令がある番地が持つ値を必要とする場合，その番地は命令の一部である．その番地をテープ3にコピーし，命令の位置をテープ1の第2トラック (図 8.22 には示されていない) を使って印付けし，必要なら命令位置に戻ってこれるようにしておく．次に，テープ1でメモリー番地を探し，それが持つ値，w とする，をテープ3にコピーする．

(4) 命令，または，テープ3の値 w を使う命令の一部を実行する．すべての可能な機械命令について調べることはしないが，w を使ってすべき種類のことには，たとえば次のようなものが考えられる．

(a) w を別の番地にコピーする．コピー先の番地を命令からとり，それをテープ3に置いて，以前議論したようにテープ1を探してその位置を見つける．コピー先の番地を見つけたら，テープ3に置いてある w を，その番地のデータ領域にコピーする．w のためにその番地の古い値が使っていた領域より多くの領域が必要だったり，より少ない領域しか必要としない場合には，使用領域を**平行移動** (shift over) によって変更する．すなわち，

i. w が置かれる場所より右にあるすべての非空白テープの内容を，メモ用テープにコピーする．

ii. w を書き込む．

iii. メモ用テープの内容をテープ1の w のすぐ右から再コピーする．

特別の場合として，コピー先として指定された番地がまだコンピュータに使用されていなくて，テープ1上に出現していないかもしれない．この場合，テープ1のそれが置かれるべき場所を見つけ，そこに平行移動で適当な空間を作り，番地と新しい値を書き込む．

(b) w を別の番地の値に加える．もう一つの番地を知るために，命令に戻り，テープ1上でその番地を見つける．その番地にある値と，テープ3にある値の2進和を実行する．二つの値を右端から順に見ていって，少し複雑だが，桁上がり伝播加算法を実行する．結果の格納により多くの領域が必要ならば，平行移動を使ってテープ1に領域を作り出す．

(c) 「飛び越し」命令．すなわち，現在テープ3にある w 番地から次の命令を取り出せ，という指令．単にテープ3をテープ2にコピーし，命令サイクルを再開する．

(5) 命令を実行し，その命令が飛び越し命令でないことを確認したら，テープ2の命令カウンタの値を1増やし，命令サイクルを再開する．

典型的なコンピュータをTMで模倣するには，つめるべき多くの詳細がまだ残っている．図8.22には入力を保持する第4テープが示されている．コンピュータはその入力列 (言語に属すか否かを検査している語) をファイルから読まなければならないが，TMは代わりにこのテープから入力を読み込む．

図8.22にはメモ用テープもある．コンピュータ命令のいくつかの模倣では，メモ用テープを有効に利用するし，乗算などには複数のメモ用テープを利用することもある．

最後に，コンピュータは入力が受理されたか否かを示す出力を出すと仮定する．この動作をテューリング機械の動作に変換するには，おそらく出力フィルに yes と書き出すような，コンピュータの関数呼び出しに対応する「受理命令」

がコンピュータにあると仮定するとよい．TM がこのコンピュータ命令の実行を模倣するときは，自身の受理状態に入り，停止する．

TM が典型的なコンピュータを模倣できることのこれまでの議論は，完璧で形式的な証明からは程遠いとはいえ，TM が，計算機ができることの正しい表現であることを，読者に納得させるのに十分なほど詳しい説明になっているだろう．したがってこれからも，任意の種類の計算装置で計算できることの形式的表現として，テューリング機械は使われ続けるだろう．

8.6.3 コンピュータとテューリング機械の計算時間の比較

さて，コンピュータを模倣するテューリング機械の計算時間の問題を扱わなければならない．前に示唆したように

- 計算時間の問題は重要である．なぜなら，TM は計算できることの検査に使われるばかりでなく，問題のコンピュータによる解決が実行可能なほどの十分な効率で計算できることの試験にも使われるからである．
- **実行可能** (tractable) **問題**，効率よく解ける問題，と**実行不能** (intractable) **問題**，解けるけれども解法を実際に使えるほど速くは解けない問題，とを分ける線は，多項式時間で計算できることと，多項式計算時間より多くかかるものとの間に引かれる．
- したがって，ある問題が，コンピュータで多項式時間で解ければ，テューリング機械でも多項式時間で解けるし，その逆も成り立つということをここで納得しておく必要がある．この多項式同値性により，テューリング機械が適当な効率でできることやできないことに関する結論は，そのままコンピュータでも成り立つ．

8.4.3 項で，1-テープ TM と多テープ TM との計算時間の差は多項式，2 次式であることを示した．したがって，コンピュータにできることなら何であっても，8.6.2 項で記述した多テープ TM が，コンピュータがかける時間の多項式時間でできることを示せば十分である．そうすれば，同じことが 1-テープ TM でも成り立つ．

上で記述したテューリング機械がコンピュータの n ステップを $O(n^3)$ 時間で模倣できることの証明を与える前に，コンピュータ命令としての掛け算の問題を片付けなければならない．問題は，コンピュータが保持できるビット数を

制限していないことから生じる．たとえば，整数2を持つ語から開始し，その語と自分自身とを掛けることを連続して繰り返すと，n ステップ後には語 2^{2^n} になる．この語を表現するには，2^n+1 ビットを必要とするので，テューリング機械はこの n ステップを模倣するのに，少なくとも n の指数関数時間必要とする．

一つの解決法は，語が固定した長さ (たとえば64ビット) を保つように強制することである．このとき，その長さを越える語を作り出す掛け算 (や他の演算) は，コンピュータを停止させ，それゆえ，テューリング機械はそれ以上模倣する必要がなくなる．しかし，ここではもう少し自由な立場をとり，コンピュータは，いくらでも長い語を取り扱うことができるが，一つの命令は，その変数より高々1ビット長い語を生成できるだけだという制限を設けることにする．＜**訳注**：ここでは，ビットがTMのテープ上での1マスを意味すると考えるとよい．＞

例 8.16　上記の制限のもとで，加算は許される．なぜなら，可算結果のビット数は可算される2数のビット数の大きい方より1ビット長いだけであるから．一方，乗算は許されない．二つの m ビット数の乗算結果の長さは $2m$ になる．しかし，m ビット数の乗算は，被乗算数の1ビット左シフト (これも長さを1ビットだけ増やす演算) を間に挟んだ m 回の可算で模倣できる．したがって，任意の長さの乗算は可能であるが，それにコンピュータが費やす時間は，被演算子の長さに比例するものと考える．

1命令当たりの1ビット増加を仮定すれば，二つの計算時間が多項式で関連付けられる．証明のアイデアは，n 命令実行後に，TMの記憶テープに貯えられた語の個数が $O(n)$ であり，各語はそれを表現するTMのマスを $O(n)$ 個必要とすることに着目することである．したがって，テープの長さは $O(n^2)$ であり，TMは一つのコンピュータ命令の実行に必要な有限個の語を，$O(n^2)$ 時間で探し出すことができる．

しかし，コンピュータ命令に対する制限がさらにもう一つ必要である．命令が，結果として長い語を生成しなくても，計算に多大な時間がかかってしまうかもしれない．それゆえ，命令それ自身が，長さ k までの語に適用されたとき，多テープテューリング機械で $O(k^2)$ のステップで実行可能であると仮定する．

実際，加算，シフト，値の比較といった典型的なコンピュータ命令は，多テープテューリング機械で $O(k)$ ステップで実行できるので，1 命令で実行可能な事柄としてはこれは十分すぎる仮定である． ▨

定理 8.17 コンピュータの命令がすべて，

(1) 語の最大長を高々1 しか増やさない

(2) 多テープ TM が長さ k の語に対して，$O(k^2)$ ステップで実行できる

という条件を満たすとき，8.6.2 項で記述したテューリング機械は，コンピュータの n ステップを $O(n^3)$ ステップで模倣できる．

証明 まず，図 8.22 の TM の 1 番目の (記憶) テープは，コンピュータのプログラムだけを持ってスタートすることに注意しよう．そのプログラムは長いかもしれないが，固定長で，コンピュータが実行するステップ数 n に依存しない．したがって，ある定数 c があって，プログラム中に現れる語や番地の最大値を表すことができる．さらに，プログラムが占める語の数を表す定数 d も存在する．

したがって，コンピュータは $c+n$ より長い語を n ステップの実行後に生み出すことはなく，それゆえ，$c+n$ ビットより長い番地を使うこともない．各命令は，値を入れる新しい番地を高々 1 個作り出すだけなので，n 命令の実行の後の番地の総数は高々$d+n$ である．番地と値の組みが TM のテープ上で占める領域は，番地やその中身とそれらを分ける区切り記号とで，高々 $2(c+n)+2$ マスだから，n ステップの模倣の後での TM のテープのマスの総数は，高々$2(d+n)(c+n+1)$ である．c と d は定数だから，このマス数は $O(n^2)$ である．

一つの命令に含まれる一定個数の番地の探索は，$O(n^2)$ で実行できることがわかっている．語の長さは $O(n)$ であるから，2 番目の仮定から，命令それ自身は TM で $O(n^2)$ 時間で実行できる．命令の実行で残る重要なコストは，TM が新たな語を保持したり，語の長さを拡張したりするためにテープ上に新たな領域を作り出すための時間である．しかし，“平行移動” は，高々$O(n^2)$ 個のデータをテープ 1 からメモテープにコピーし，またテープ 1 に戻すことからなる．したがって，“平行移動” は 1 命令あたり $O(n^2)$ 時間でできる．

結局，TM はコンピュータの 1 命令を $O(n^2)$ ステップで模倣できる．したがって，定理の主張にあるように，コンピュータの n 命令は，テューリング機械の $O(n^3)$ ステップで模倣される． ▨

さて，最終的に，ステップ数を3乗すれば多テープTMがコンピュータを模倣できることがわかった．8.4.3項で，1-テープTMは，ステップ数の2乗により，多テープTMを模倣できることがわかっている．したがって，次の定理が成り立つ．

定理 8.18 定理8.17で述べたタイプのコンピュータの n ステップ命令は，1-テープテューリング機械で高々$O(n^6)$ ステップで模倣される．

8.7 第8章の要約

- **テューリング機械**：TMは現実のコンピュータや計算に関する他の数学的なモデルと同等の能力を持つ抽象的な計算機械である．TMは有限状態制御部とマスに分かれた無限の長さテープから成る．各マスは有限個のテープ記号の一つを持ち，テープヘッドはどれかのマスに位置する．TMは，現在の状態とテープヘッドが見ているマスにある記号に基づいて動作する．1回の動作で，TMは状態を変え，見ているマスにテープ記号を上書きし，ヘッドを1マス右か左に動かす．
- **テューリング機械による受理**：TMは，有限長のテープ記号列をテープ上に入力として持ち，残りはすべて空白のマスからスタートする．空白はテープ記号の一つで，入力は，入力記号と呼ばれる空白を含まないテープ記号の部分集合から選ばれる．TMは受理状態に入ると入力を受理する．
- **帰納的可算言語**：TMで受理される言語は帰納的可算言語(RE)と呼ばれる．したがって，いかなる種類の計算装置でも，認識または受理される言語はRE言語である．
- **TMの時点表示**：TMの現在の状況を，テープの内容(非空白部分)を含む有限長の記号列で表すことができる．TMの状態とテープヘッドの位置は，テープを表す記号列中の現在ヘッドが見ている記号の左隣に状態を挿入することによって表される．
- **有限制御部への記憶**：与えられた言語を受理するTMを設計するとき，状態が二つ以上の成分から成ると考えると都合のよいことがある．成分の一つは制御成分で，通常の状態が持つ機能を果たす．他の成分はTMが覚えておくべきデータを保持する．
- **多重トラック**：テープ記号が固定された数の成分を持つベクトルであると考えると便利なことが多い．各成分を，テープのトラックに分けて図示することができる．
- **多テープテューリング機械**：拡張されたTMモデルで，固定された数(2個以上)のテープを持つ．このタイプのTMの動作は，状態と各テープのヘッドが読んでいる記号から成るベクトルとによって定まる．1動作で，多テープTMは状態を変え，各テープヘッドが位置するマスに記号を上書きし，テープヘッドの全部または一部

をそれぞれ独立に右または左に1マス動かす．ある種の言語を通常の1-テープ機械より速く認識することはできるが，多テープTMが機能的可算でない言語を受理することはできない．

◆**非決定性テューリング機械**：NTM (非決定性TM) は，状態と読み込み記号に対する動作 (新しい状態，上書き記号，ヘッドの動きの三つ組) に有限個の選択肢がある．入力を受理するのは，受理状態を持つID (時点表示) にNTMを導くような，一連の選択肢があるときである．一見，決定性TMより能力が高いように思われるが，NTMもREでない言語を認識することはできない．

◆**半無限テープテューリング機械**：TMのテープを無限に延ばせる方向を右方向だけにし，ヘッドの初期位置より左にはマスがないように制限することができる．そのようなTMは任意のRE言語を受理できる．

◆**多スタック機械**：多テープTMの各テープをスタックとして振舞うように制限できる．入力は別のテープにあり，そのテープは有限オートマトンやPDAと同様に左から右へ一度読まれるだけである．1-スタック機械は実際はDPDAであり，2-スタック機械は任意のRE言語を認識できる．

◆**カウンタ機械**：多スタック機械のスタックを，さらに底記号以外に1記号しか扱えないように制限することができる．したがって，スタックの機能は，カウンタとして非負整数を保持し，その値が0か否かを検査できる，というだけである．2-カウンタ機械であれば，任意のRE言語を認識できる．

◆**コンピュータによるテューリング機械の模倣**：原理的には，現実のコンピュータでTMを模倣することができる．ただし，TMテープの (無限の) 空白部分を模倣するために，ディスクのような取替え可能な記憶媒体が，必要に応じて無限に供給され得るものと仮定する．ディスクを作成するための原料が無限に存在するわけではないから，この議論には疑問が残る．しかし，宇宙全体にどのくらいの記憶装置や媒体が存在するかの限界は知られていないし，疑いもなく莫大なので，この無限供給の仮定は，TMにおける無限テープ同様，実際問題としては現実的であり，一般的に受け入れられている．

◆**テューリング機械によるコンピュータの模倣**：TMは，コンピュータのレジスタや主記憶，ディスク，その他の記憶装置の内容や場所を1本のテープに保持することによって，その記憶や制御を模倣できる．したがって，TMにできないことは，現実のコンピュータでもできないと確信できる．

8.8 第 8 章の参考文献

テューリング機械は [8] による．ほとんど同じころ，計算できることとは何かを特徴付けるより機械的形式をとらない提案が，A. Church [1]，S. C. Kleene [5]，E. Post [7] 等によりなされた．これらすべてに先立って，K. Gödel [3] の仕事があり，事実上，コンピュータがすべての数学的問題を (自動的に) 解決するような方法は存在しないことを示した．

多テープテューリング機械の研究，特に 1-テープ TM との計算時間の比較，は Hartmanis と Stearns[4] で始められた．多スタック機械とカウンタ機械の考察は [6] によるが，ここでの構成法は [2] による．

テューリング機械の受理や停止の概念の代わりに，“hello, world” プログラムを使う 8.1 節の方法は，S. Rudich の未出版のノートによる．

[1] A. Church, “An undecidable problem in elementary number theory,” *American J. Math.* **58** (1936), pp.345–363.

[2] P. C. Fischer, “Turing machines with restricted memory access,” *Information and Control* **9**:4 (1966), pp.364–379.

[3] K. Gödel, “Über formal unentscheidbare sätze der Principia Mathematica und verwander systeme,” *Monatschefte für Mathematik und Physik* **38** (1931), pp.173–198.

[4] J. Hartmanis and R. E. Stearns, “On the computational complexity of algorithms,” *Transactions of the AMS* **117** (1965), pp.285–306.

[5] S. C. Kleene, “General recursive functions of natural numbers,” *Mathematische Annalen* **112** (1936), pp.727–742.

[6] M. L. Minsky, “Recursive unsolvability of Post’s problem of ‘tag’ and other topics in the theory of Turing machines,” *Annals of Mathematics* **74**:3 (1961), pp.437–455.

[7] E. Post, “Finite combinatory processes-formulation,” *J. Symbolic Logic* **1** (1936), pp.103–105.

[8] A. M. Turing, “On computable numbers with an application to the Entscheidungsproblem,” *Proc. London Math. Society* **2**:42 (1936), pp.230–265. See also *ibid.* **2**:43, pp.544–546.

9 決定不能性

8.1 節で行った，コンピュータで解けない問題の存在に対するもっともらしい議論を，テューリング機械上で繰り返すことから始めよう．8.1 節の “証明” が持つ問題は，C (や他のプログラミング言語) が現実のコンピュータ上で実現されるときに必ず生じる限界を無視せざるを得ない点にあった．しかし，番地空間のサイズなどのこういった限界は，重要な問題ではない．むしろ，コンピュータの進歩につれて，番地空間のサイズや主記憶のサイズ，その他の指標は，年とともにいくらでも増大すると期待できる．

テューリング機械に焦点をあてればこれらの限界は存在しないので，計算装置なるものが，理想的な状況で達成できる事柄の本質をよりよく捉えられる．この章ではテューリング機械に解けない問題が存在することの形式的な証明を与える．8.6 節でテューリング機械が，現実のコンピュータには存在する上記の制限を持たない理想のコンピュータでさえ模倣できるということを示したので，次に述べる問題ではコンピュータでは解けない —— たとえ上記の制限をどれだけ緩めたとしても解けない —— ということに対する厳密な議論を展開できる．

- このテューリング機械は自分自身 (のコード) を入力として受理するか

次に，テューリング機械で解ける問題を二つのクラスに分ける．**アルゴリズム** (algorithm) (すなわち，入力の受理如何に関わらず停止するテューリング機械) を持つ問題と，受理されない入力に対しては無限に動き続ける可能性のあるテューリング機械でしか解けない問題とである．後者の受理形式は，テューリング機械をいくら走らせても入力が受理されるか否かが判明しないことがあるという意味で，問題が多い．したがってこの章では，一部の入力に対して停止しないテューリング機械で受理されるか否かよりむしろ，アルゴリズムを持たない「決定不能問題」を証明する技法を主に議論する．

次の問題が決定不能であることを示す．

- 与えられたテューリング機械は与えられた入力を受理するか．

この問題の決定不能性を利用して，他の多くの決定不能問題を示そう．テューリング機械が受理する言語に関する自明でない問題はすべて決定不能であるという結果とともに，テューリング機械やプログラム，コンピュータ等にまったく関係のない多くの問題の決定不能性が示される．

9.1 帰納的可算でない言語

言語 L が**帰納的可算** (recursively enumerable, RE) であるというのは，$L = L(M)$ となる TM M が存在するときであった．9.2 節では「帰納的」あるいは「決定可能」な言語を紹介する．それらは帰納的可算であるだけでなく，入力の受理・非受理に関係なく常に停止するテューリング機械で受理される言語である．

最終目標は，次の対 (M, w) から成る言語の決定不能性を示すことである．

(1) M はアルファベット $\{0,1\}$ 上のテューリング機械で，2 進 ($\{0,1\}$) で適切に符号化される．

(2) w は $\{0,1\}$ 上の列である．

(3) M は w を受理する．

入力が 2 進アルファベットに制限されたこの問題が決定不能ならば，任意のアルファベット上の TM に対するより一般的な問題は，明らかに決定不能である．

最初のステップは，この問題を特定の言語の所属問題として設定することである．したがって，TM の状態数に関わらず，0 と 1 だけを使って TM を符号化しなければならない．この符号化が得られれば，任意の 2 進列を TM とみなすことができる．2 進列が TM の正しい符号 (表現形式) になっていない場合，それは動作をまったく持たない TM だと思えばよい．

この節で達成する中間目標には，「対角言語」L_d が関係する．L_d は，列 w で表された TM が w を受理しない，そのような w の集合である．L_d を受理する TM は存在しないことを示そう．ある言語を受理する TM が存在しないということは，その言語が決定不能である (すなわち，アルゴリズムや常に停止する TM が存在しない) ことよりずっと強いことを意味する．

言語 L_d は，8.1.2 項で仮定したプログラム H_2 に似た役割をする．H_2 は，入力として与えられたプログラムが hello, world と表示しないときに，hello, world と表示した．自分自身を入力させると矛盾に陥るために H_2 は存在でき

なかったように，L_d を受理するテューリング機械は自分自身の符号を入力させると結果が合わなくなるため存在できない．

9.1.1　2 進文字列の数え上げ

以下では，非負整数と 2 進列を 1 対 1 に対応させる．2 進列 w に対し，$1w$ が表す 2 進数 i を対応させ，i 番目の列と呼ぶ．したがって，ε は 1 番目で，0 が 2 番目，1 が 3 番目，00 が 4 番目，01 が 5 番目…となる．同じことだが，列を長さの順に並べ，同じ長さの列は辞書式順序で並べてもよい．これ以降，i 番目の列を w_i と書く．

9.1.2　テューリング機械の符号化

次の目標は，入力アルファベット $\{0,1\}$ 上のテューリング機械自身を 2 進列とみなすことができるような 2 進符号を工夫することである．そうすれば，前節で 2 進列を数え上げる方法を定めたところなので，テューリング機械を非負整数に対応させて「i 番目のテューリング機械 M_i」といういい方ができる．TM $M = (Q,\ \{0,1\},\ \Gamma,\ \delta,\ q_1,\ B,\ F)$ を 2 進列で表すために，最初に状態とテープ記号とヘッドの移動方向 L と R を整数に対応させる．

- 状態は，r を状態数として，$q_1, q_2, \ldots, q_r$ であるとする．開始状態は q_1 で，受理状態は q_2 のみであるとする．TM は受理状態に入ると停止すると仮定してよいので，二つ以上の受理状態は必要ない．
- テープ記号は，記号数を s として，$X_1, X_2, \ldots, X_s$ であるとする．X_1 は常に 0 であり，X_2 は 1，X_3 は空白 B であるとする．しかし，他のテープ記号は残りの整数に適当に割り当てられる．
- 方向 L は D_1，R は D_2 で表す．

TM の状態やテープ記号へは，多くの異なる順番で整数を割り当てることができるので，TM の符号化は何通りもあり得る．しかしこのことは，以下で $L(M) = L_d$ となる TM M を表す 2 進符号が存在しないことを示す際には，重要でない．

状態やテープ記号，方向を表す整数を定めたので，次に遷移関数 δ を符号化しよう．遷移規則 $\delta(q_i, X_j) = (q_k, X_l, D_m)$ があるとする．この規則を列 $0^i10^j10^k10^l10^m$ で符号化する．i, j, k, l, m はすべて 1 以上なので，この符号内で 1 が二つ以上連続することはない．

TM M 全体を表す符合は，遷移規則を表す符合 $C_1, C_2, \ldots, C_n$ を列 11 で区切って適当な順番で並べた $C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$ である．

例 9.1 TM $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$ を考えよう．δ は次の規則から成る．

$$\delta(q_1, 1) = (q_3, 0, R)$$

$$\delta(q_3, 0) = (q_1, 1, R)$$

$$\delta(q_3, 1) = (q_2, 0, R)$$

$$\delta(q_3, B) = (q_3, 1, L)$$

各規則を表す符合はそれぞれ，

0100100010100

0001010100100

00010010010100

0001000100010010

となる．たとえば，最初の規則は，$1 = X_2$，$0 = X_1$，$R = D_2$ なので $\delta(q_1, X_2) = (q_3, X_1, D_2)$ と書け，その 2 進符号は上に示したように $0^1 10^2 10^3 10^1 10^2$ となる．M 全体を表す符号は，次の通りである．

01001000101001100010101001001100010010010100110001000100010010

M を表す 2 進符号は他にも多くある．特に，四つの規則を並べる順番は任意なので，それだけでも $4! = 24$ 通りある． ▨

9.2.3 項では，TM と列の対 (M, w) を符号化する必要がある．この対の符号は，M の符号と w とを列 111 で区切って並べたものである．TM を表す正しい符号は，列 111 を含まないので，最初に現れる 111 が M の符号と w との区切りとなる．たとえば，M が例 9.1 の TM で w が 1011 であったとすれば，(M, w) の符号は例 9.1 の符号の後に 1111011 をつなげた列である．

9.1.3 対角線言語

9.1.2 項で TM を 2 進符号化できたので,「i 番目のテューリング機械」M_i の具体的な意味が定義できる．すなわち，M_i はその符号が i 番目の 2 進列 w_i であるような TM である．どんな TM にも対応しない正整数が数多くある．たと

えば，11001 は 0 から始まらないし，0010111010010100 は三つの連続した 1 を含むので，どちらも TM の符号ではない．w_i が TM の正しい符号になっていない場合，M_i は 1 個の状態を持ち遷移が何もない TM であるとする．すなわち，これらの i に対応する M_i は，任意の入力に対しそれを受理することなくただちに停止する．したがって，w_i が TM の正しい符号でなければ，$L(M_i) = \varnothing$ である．

さて，重要な定義をしよう．

- **対角線言語** (diagonalization language)L_d は，$w_i \notin L(M_i)$ を満たす列 w_i の集合である．

すなわち，L_d は自分自身の符号 w を入力として受理しないような TM M の符号 w の全体である．

L_d が "対角線" 言語と呼ばれる理由は，図 9.1 を考えるとわかるだろう．この表はすべての i, j に対し，TM M_i が w_j を受理するか否かを表し，1 は受理を，0 は非受理を意味する[1]．第 i 行は言語 $L(M_i)$ の**特徴ベクトル** (characteristic vector) とみなせる．すなわち，行中の 1 は，対応する列がこの言語の元であることを意味する．

図 9.1 の対角線成分は，M_i が w_i を受理するか否かを表す．L_d を構成するには対角線成分の補数をとる (0 と 1 を交換する)．たとえば，図 9.1 が正しい表

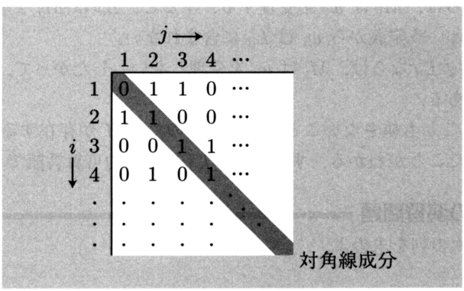


図 9.1　テューリング機械による列の受理を表す表

[1] 実際の表はここで例示したものとは大きく異なる．小さな i の値は TM の正しい符号になっていないので，動作を持たない単純な TM を表す．したがって，表の最初の方の行はすべて 0 であるべきだろう．

だったとすると，補数をとった対角線成分は $1, 0, 0, 0, \ldots$ で始まる．したがって L_d は $w_1 = \varepsilon$ を含み，$w_2 = 0, w_3 = 1, w_4 = 00$ を含まない．

対角線成分の補数をとって，どの行にも現れない言語の特徴ベクトルを作る技法は，**対角線論法** (diagonalization) と呼ばれる．対角線成分の補数はそれ自身言語 L_d の元を定義する特徴ベクトルである．このベクトルは図 9.1 のどの行とも少なくとも 1ヶ所異なる．したがって，対角線成分の補数ベクトルは，TM で受理されるどの言語の特徴ベクトルにもならない．

9.1.4 L_d が帰納的可算でないことの証明

前節で議論した，特徴ベクトル列とその対角線成分に関する直観にしたがって，L_d を受理するテューリング機械は存在しないという，基本的な結果を形式的に証明しよう．

定理 9.2 L_d は帰納的可算言語ではない．すなわち，L_d を受理するテューリング機械は存在しない．

証明 TM M が L_d を受理できたとする．L_d はアルファベット $\{0,1\}$ 上の言語であるから，M は上で構成した，$\{0,1\}$ を入力とするすべてのテューリング機械のリストの中に含まれる．したがって，M の (2 進) 符号が少なくとも一つ存在するので，それを i としよう．すなわち，$M = M_i$ とする．

さて，$w_i \in L_d$ だろうか．

- $w_i \in L_d$ ならば，M_i は w_i を受理する．すると，L_d は M_j で受理されない w_j だけを含むという定義から w_i は L_d に含まれない．
- 同様に，$w_i \notin L_d$ ならば，M_i は w_i を受理しない．したがって，L_d の定義から，$w_i \in L_d$ である．

w_i は L_d に属すことも属さないこともできないので，M が存在するという最初の仮定に矛盾があったことがわかる．すなわち，L_d は帰納的可算言語ではない． ▨

9.1.5 9.1 節の演習問題

♣ **問 9.1.1** 次の列を求めよ．

*a) w_{37}

b) w_{100}

♣ **問 9.1.2** 図 8.9 のテューリング機械の 2 進符号を一つ求めよ．

♣ **!問 9.1.3** 以下に L_d と微妙に異なる定義を持つ言語を定義する．対角線論法を用いて，それらがテューリング機械で受理されないことを示せ．証明には，図 9.1

の行列上で，対角線そのものではなく，別の無限点列を見つけなければならない．

*a)　M_{2i} で受理されない w_i の集合．

b)　w_{2i} が M_i で受理されないような w_i の集合．

♣ **!問 9.1.4**　これまでは，入力アルファベット $\{0,1\}$ 上のテューリング機械だけを考えてきた．入力アルファベットに関わらずすべてのテューリング機械に番号を付けたいとする．それは完全に可能だというわけではない．なぜなら，入力記号以外のテープ記号や状態名は任意に定められるが，入力記号はそうはいかないからである．たとえば，言語 $\{0^n1^n \mid n \geq 1\}$ と $\{a^nb^n \mid n \geq 1\}$ は，ある意味で似てはいるが，同じ言語ではなく，異なる TM で受理される．しかし，TM の入力アルファベットがそこから選ばれる無限集合 $A = \{a_1, a_2, \ldots\}$ を仮定しよう．A の有限部分集合を入力アルファベットとするすべての TM を番号付ける方法を示せ．

9.2　帰納的可算な決定不能問題

それを受理するテューリング機械を持たない問題 —— 対角線言語 L_d —— があることがわかった．次の目標は，TM で受理される帰納的可算 (RE) 言語のクラスを二つに区分することである．一つは，通常アルゴリズムと考えられるものに対応するクラスで，このクラスの言語に対しては，単に受理するだけでなく，入力列が言語に属さない場合にはそのことも教えてくれる TM がある．このような TM はすべての入力に対して，受理状態に達するか否かに関わらず，いつかは停止する．

二つ目のクラスは，停止性が保証されたどの TM でも受理できない RE 言語から成る．このクラスの言語は不便なやり方で受理される．入力がその言語に属せば，いつかはそのことがわかる．しかし入力がその言語に属さなければ，TM は永久に走り続けるかもしれない．その場合いつまでたっても，入力が受理されないということを知る手立てはない．このタイプの言語の一例は，後にわかるように，TM M の 2 進符号とそれが受理する列 w の対 (M, w) の集合である．

9.2.1　帰納的言語

言語 L が**帰納的** (recursive) であるというのは，次の条件を満たすテューリング機械 M で受理されるときである．

(1) $w \in L$ ならば，M は受理状態に入って停止し，

(2) $w \notin L$ ならば，M は受理状態に入ることなく停止する．

このタイプのTMは，正しく定義され常に正しい答を出すステップ列という，アルゴリズムの直観的な概念に適合する．よくやるように言語 L を“問題”と考えれば，問題 L は帰納的であるとき**決定可能** (decidable) であるといい，帰納的でないとき**決定不能** (undecidable) であるという．

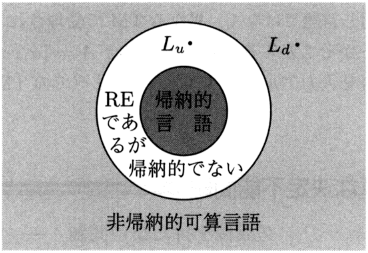


図 9.2 帰納的言語，RE 言語，非帰納的可算言語の関係

ある問題を解くアルゴリズムが存在するか否かは，その問題を解くテューリング機械が存在するか否かより重要なことが多い．以前述べたように，停止性が保証されないTMからはある列が言語に属さないと結論するのに十分な情報が得られないかも知れないので，“問題が解ける”とはいいがたい．したがって，言語や問題を決定可能 (アルゴリズムで解かれる) と決定不能とに分けることは，帰納的可算言語 (TMで受理されるもの) とそうでないものとに分けることより，重要なことが多い．図 9.2 は以下の三つの言語クラスの間の関係を図示したものである．

(1) 帰納的言語

(2) 帰納的ではないが帰納的可算である言語

(3) 非帰納的可算 (非 RE) 言語

図 9.2 には非帰納的可算言語 L_d と，これから RE ではあるが帰納的でないことを示す“万能言語”L_u の位置を示してある．

なぜ「帰納的」というのか

現代のプログラマーは帰納的関数 (recursive function) をよく知っている．にも関わらず，帰納的関数と常に停止する TM とはほとんど関係がないように思われている．さらに悪いことに，その否定 —— 非帰納的 (nonrecursive)，あるいは決定不能 —— はどんなアルゴリズムでも認識されない言語を意味するが，非再帰的 (nonrecursive) は，関数の再帰呼び出し (recursive call) を必要としないほど単純な計算を意味する言葉としてよく知られている．

述語「帰納的」は「決定可能」と同義語であり，由来はコンピュータの出現以前の数学にさかのぼる．当時は，(繰り返しやループでなく) 再帰 (帰納) に基づく計算概念の形式化が一般的に使われていた．ここでは説明しないが，この概念は，LISP や ML といった関数型言語の趣を持っていた．その意味では，問題が「帰納的」であるということは,「それを解く帰納的関数を書けるほど十分単純な問題であり，関数は常に定義されている」という肯定的な意味を持つ．それは今日，テューリング機械に関連して使われている意味と同じである．

「帰納的可算 (recursively enumerable)」という述語も同じ語源を持つ．関数は言語に属するすべての列をある順序で並べる，すなわち，数え上げる (enumerate) ことができる．それに属する列をある順序で並べることのできる言語のクラスは，一般の受理しない列上では無限の動くこともある TM で受理される言語のクラスと等しい．

＜**訳注：**日本語では上にあるように “recursive” にあたる概念を場合に応じて「帰納的」と「再帰的」で使い分けることが多い．＞

9.2.2 帰納的言語と帰納的可算言語の補集合

言語が図 9.2 のリングに属する (すなわち RE であるが帰納的でない) ことを示すときの強力な道具は，言語の補集合を考えることである．まず帰納的言語は補集合演算で閉じていることを示そう．そうすれば，言語 L が RE で，しかも L の補集合 $\overline{L}$ が RE でなければ，L は帰納的でないことがわかる．というのは，もし L が帰納的なら，$\overline{L}$ も帰納的であり当然 RE であるから．では，帰納的言語のこの重要な閉包性を証明しよう．

定理 9.3 L が帰納的言語ならば，$\overline{L}$ も帰納的である．

証明 L は常に停止する TM M で受理されるとする．図 9.3 に示した構成法で，$\overline{L} = L(\overline{M})$ となる TM $\overline{M}$ を構成する．すなわち，$\overline{M}$ は M と同じように動作するが，

以下のように修正される．

(1) M の受理状態は $\overline{M}$ の非受理状態になり，そこからの遷移は定義されない．すなわち，これらの状態では，$\overline{M}$ は入力を受理せずに停止する．

(2) $\overline{M}$ は新しい受理状態 r を持ち，r からの遷移は持たない．

(3) 遷移を持たないような M の非受理状態とテープ記号のすべての対 (すなわち，M が受理せずに停止する対) に対し，受理状態 r への遷移を追加する．

M は停止性が保証されているので，$\overline{M}$ も停止性が保証される．さらに $\overline{M}$ は，M が受理しない列，それだけを受理するので，$\overline{L}$ を受理する． ▨

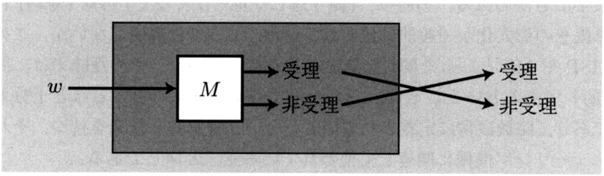


図 9.3 帰納的言語の補集合を受理する TM の構成

言語の補集合に関する重要な結果がもう一つある．それは図 9.2 において，ある言語とその補集合とが占める位置を，以下の定理に述べるように制限する．

定理 9.4 言語 L とその補集合 $\overline{L}$ がともに RE ならば，L は帰納的である．したがって，定理 9.3 により，$\overline{L}$ も帰納的である．

証明 証明のアイデアを図 9.4 に示した．$L = L(M_1)$, $\overline{L} = L(M_2)$ とする．M で M_1 と M_2 を並列に模倣する． 模倣を容易かつわかりやすくするために M を 2-テープ TM として構成し，必要なら 1-テープ TM に変換する．M の片方のテープは M_1 のテープを模倣し，他方は M_2 のテープを模倣する．さらに，M_1 の状態と M_2 の状態の対が M の状態である．

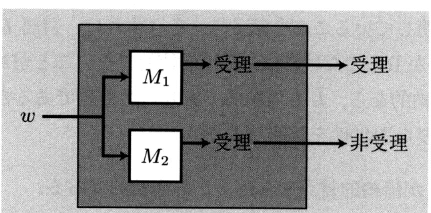


図 9.4 ある言語とその補集合を受理する二つの TM の模倣

入力列 w が L に属せば，M_1 はいつか w を受理し，M も受理して停止する．w が L に属さなければ，$\overline{L}$ に属すので M_2 が受理し，この場合 M は受理せずに停止する．したがって，M はすべての入力に対し停止し，$L(M) = L$ であるから，L は帰納的である． ▨

定理 9.3 と 9.4 は次のようにまとめることができる．図 9.2 における言語 L とその補集合 $\overline{L}$ の配置は 9 通りあるが，次の 4 通りだけが実際に起こり得る．

(1) L と $\overline{L}$ がともに帰納的，すなわち中央の円内に属する．

(2) L も $\overline{L}$ も RE でない，すなわちともにリングの外にある．

(3) L は RE であるが帰納的でない．この場合 $\overline{L}$ は RE でなく，L はリングの中にあり，$\overline{L}$ はリングの外にある．

(4) $\overline{L}$ は RE であるが帰納的でない．この場合 L は RE でなく，(3) の L と $\overline{L}$ を交換した形になる．

定理 9.3 から，L と $\overline{L}$ の一方が帰納的で他方が帰納的でない二つの場所に位置する可能性が除かれる．定理 9.4 は，L と $\overline{L}$ がともに RE でかつ帰納的でない (つまりリングに位置する) 可能性を排除する．

例 9.5 例として RE でない言語 L_d を考えよう．$\overline{L_d}$ は帰納的でない．しかし，$\overline{L_d}$ は RE でないかも知れないし，帰納的でない RE 言語かも知れない．そして実際，後者なのである．

$\overline{L_d}$ は M_i で受理される w_i の集合である．これは次項 9.2.3 で RE であることが証明される万能言語 $L_u = \{(M, w) \mid w \in L(M)\}$ と似た定義を持ち，実際，L_u の証明と同様にして，$\overline{L_d}$ が RE であることが示せる． ▨

9.2.3 万 能 言 語

テューリング機械を使って任意のプログラムを搭載したコンピュータを模倣する方法はすでに 8.6.2 項で非形式的に議論した．いい換えれば，TM を，プログラムとデータとが入力として一つ以上のテープに与えられる「プログラム内蔵方式のコンピュータ」として構成した．この項ではそのアイデアを，内臓プログラム方式をテューリング機械で実現することから得られる厳密さを加えて，より形式的に繰り返そう．

万能言語 L_u は，$\{0,1\}$ 上の TM M と $L(M)$ に属する w の対 (M, w) の (9.1.2 項の記法による) 2 進符号の集合である．すなわち，L_u は TM とそれ

で受理される入力列の対を表す列の集合である．一般に**万能テューリング機械** (universal Turing machine) と呼ばれる，L_u を受理する TM U が存在することを示そう．U の入力は 2 進列であるから，U は実際，9.1.2 項で構成した 2 進入力 TM のリスト中のある M_j である．

U は図 8.22 にならって，多テープ TM で説明するのが一番楽である．U では，最初テープ 1 に (M, w) の 2 進符号が置かれる．テープ 2 は M のテープを，M の符号化と同じ形式で保持するために使われる．すなわち，M のテープ記号 X_i は列 0^i で表され，各記号は列 1 で区切られる．U のテープ 3 は M の状態を保持し，状態 q_i を 0^i で表す．U の概要を図 9.5 に示した．

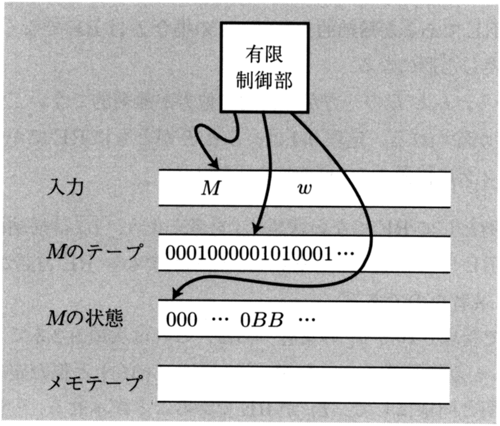


図 9.5 万能テューリング機械の構成

U の動作は次のようにまとめることができる．

(1) M の符号が本当に正しい TM の符号になっていることを確認する．TM の符号でなければ，U は入力を受理することなく停止する．正しくない符号は動作を持たない TM を表すと仮定していて，そのような TM は入力を何も受理しないので，U のこの動作は正しい．

(2) テープ 2 に w を符号化した形で置く．すなわち，w 中の各記号 0 に対し列 10 をテープ 2 に置き，w 中の各 1 に対して，対応する場所に列 100 を置く．模倣される M の空白記号は 1000 で表されるが，最初はテープ

2 に現れず，w を表す部分以外は U の空白記号を保持する．しかし，M の空白を探しているときに U が自身の空白記号に出会ったら，U はその空白を M の空白記号を表す列 1000 に置き換える．

(3)　テープ 3 に M の開始状態を表す列 0 を置き，U の第 2 ヘッドを入力 w の符号を保持する最初のマスに置く．

(4)　M の動作を模倣するために，テープ 3 上の状態が 0^i でテープ 2 のヘッド位置から始まる 0 の列が 0^j のとき，U はテープ 1 上で遷移 $0^i10^j10^k10^l10^m$ を探す．この遷移は，M が次に行う動作を表し，U は

(a)　テープ 3 の内容を 0^k に置き換えて M の状態遷移を模倣する．それには，まずテープ 3 の 0 をすべて空白に置き換え，次に 0^k をテープ 1 からテープ 3 にコピーする．

(b)　テープ 2 の 0^j を 0^l に置き換える．これは M のテープ記号の書き換えに対応する．0^l が占める領域が前と異なる (つまり $0^i \neq 0^l$) とき，メモテープと (8.6.2 項の) 平行移動技法を使って必要な領域を作り出す．

(c)　テープ 2 のヘッドを，移動方向を表す m の値に応じて，左 ($m = 1$) または右 ($m = 2$) の 1 のマスに移動させる．こうして，U は M のヘッドの動きを模倣する．

(5)　M が受理状態に入れば，U も受理状態に入る．

このようにして，U は w 上の M の動作を模倣する．U が符号 (M, w) を受理する必要十分条件は M が w を受理することである．

より効率的な万能 TM

平行移動を必要としない，U による M のより効率的な模倣では，U は最初に M が使うテープ記号の数を決定する．それが $2^{k-1}+1$ 以上 2^k 以下ならば，テープ記号を k-ビットの 2 進符号で一通りに表現できる．そして M のテープの 1 マスは U の連続した k マスで表される．事態をさらに簡単にするために，M の遷移は U によって，本文で使っていた可変長 1 進符号でなく，ここで導入した固定長 2 進符号を使う形に書き換えられる．

9.2.4 万能言語の決定不能性

RE でかつ帰納的でない言語を示そう．言語 L_u である．L_u が決定不能である (つまり帰納的でない) という知識は L_d が RE でないという知識よりいろいろな意味で価値がある．実際，L_u の他の問題 P への還元からは P が RE か否かに関わらず P の決定不能性が導かれる．しかし，L_d の P への還元は P が RE でないときにのみ可能であり，RE である問題の決定不能性の証明には使えない．一方，問題が RE でないことの証明には RE である L_u は役に立たず L_d が使われる．

定理 9.6 L_u は RE であるが，帰納的でない．

証明 前項 9.2.3 で L_u が RE であることを証明した．L_u が帰納的だとしよう．すると定理 9.3 より，L_u の補集合 $\overline{L_u}$ も帰納的になるはずである．しかし，$\overline{L_u}$ を受理する TM があれば，L_d を受理する TM を以下に説明する方法で作ることができてしまう．L_d が RE でないことはすでにわかっているので，L_u が帰納的であるという仮定が間違っていたことになる．

$L(M) = \overline{L_u}$ としよう．図 9.6 に示したように，M を次のように修正して，L_d を受理する M' が構成できる．

(1) M' は与えられた入力 w を，列 $w111w$ に変更する．練習問題として，このステップを行う 1-テープ TM のプログラムを直接書いてみるとよい．しかし，より簡単には，第 2 テープを w のコピーに使う 2-テープ TM を構成して，それを 1-テープ TM に変換すればよい．

(2) M' は新しい入力 $w111w$ 上で M を模倣する．w が i 番目の列 w_i であれば，M' は M_i が w_i を受理するか否か決定する．M は $\overline{L_u}$ を受理するのであるから，M' が w_i を受理する必要十分条件は M_i が w_i を受理しないこと，すなわち $w \in L_d$ である．

停止問題

L_u 同様，RE であるが帰納的でない問題として TM の停止問題が有名である．実際，テューリングによるもとのテューリング機械は，受理状態による受理でなく停止による受理で定義されていた．TM M に対し，M が (受理・非受理に関わらず) 停止する入力列 w の集合を $H(M)$ とする．このとき，**停止問題** (halting problem) は，$w \in H(M)$ である対 (M, w) の集合である．この問題/言語は RE であるが帰納的でない言語のもう一つの例である．

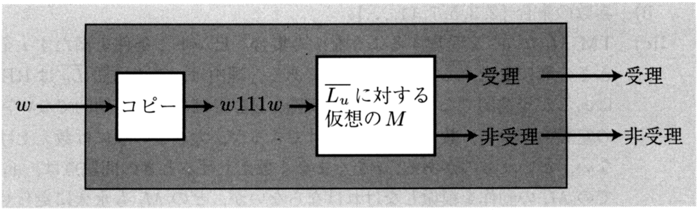


図 9.6 L_d の $\overline{L_u}$ への還元

したがって，$L(M') = L_d$ であり，定理 9.2 よりそのような M' は存在しないから，L_u は帰納的でないと結論できる． ▨

9.2.5 9.2 節の演習問題

♣ **問 9.2.1** 停止問題，すなわち，M と M が停止する入力 w の対 (M, w) の集合は RE かつ非帰納的であることを示せ．

♣ **問 9.2.2** 9.2.1 項の囲み記事 "なぜ帰納的か" で，計算できることのモデルとしてテューリング機械と対等な「帰納的関数」という概念があると述べた．この問では，帰納的関数の記法の一例を示そう．**帰納的関数** (recursive function) は有限個の規則で定義される関数 F である．規則は関数 F の値を，変数，非負整数定数，次者 (+1) 関数，関数 F 自身，を使って組み立てられる式によって定める．たとえば，**アッカーマン** (W. Ackermann) **関数** (Ackermanns's function) は次の規則で定められる．

(1) 任意の $y \geq 0$ に対し，$A(0, y) = 1$.

(2) $A(1, 0) = 2$.

(3) 任意の $x \geq 2$ に対し，$A(x, 0) = x + 2$.

(4) 任意の $x \geq 0$ と $y \geq 0$ に対し，$A(x+1, y+1) = A(A(x, y+1), y)$.

次の問に答えよ．

*****a) $A(2, 1)$ を求めよ．

!b) $A(x, 2)$ は x のどんな関数か．

!c) $A(4, 3)$ を求めよ．

♣ **問 9.2.3** 多テープテューリング機械 M が非負整数の集合 $\{i_1, i_2, \ldots\}$ を**数え上げる** (enumerate) というのは，すべてのテープが空白の状況から出発してテープの一つに列 $10^{i_1}10^{i_2}\ldots$ を書き出すときであるとする．次の集合を数え上げる TM を非形式的に述べよ．

*****a) 平方数の集合 $\{1, 4, 9, \ldots\}$.

b) 素数の集合 $\{2, 3, 5, 7, 11, \ldots\}$.

!!c) TM M_i が w_i を受理するような i の集合．**ヒント**：条件を満たす i を大きさの順に生成することは不可能である．理由は，この言語 $\overline{L_d}$ は RE ではあるが帰納的でない点にある．実際，RE ではあるが帰納的でない言語の定義の一つに，数え上げることはできるが，大きさの順には数え上げられない，というものがある．i をくまなく数え上げるときの問題点は，w_i 上での M_i の動作を模倣しなければならないが，どの M_i も永久に走らせるわけにはいかないという点にある．さもないと，停止しない (M_i, w_i) の模倣を始めたとたんに，他の (M_j, w_j) $(i \neq j)$ の模倣を排除してしまうことになる．したがってこれらを巡回的に処理し，第 k ラウンドでは有限個の (M_i, w_i) の有限ステップまでを模倣するようにしなければならない．こうすれば，各ラウンドは有限時間で終了する．各 (M_i, w_i) と各ステップ数 s について，(M_i, w_i) を s ステップまで模倣するラウンドの存在が保証される限り，いつかは w_i を受理する M_i がみつかり，i を書き出すことができる．

♣ * **問 9.2.4** $L_1, L_2, \ldots, L_k$ をアルファベット Σ 上の言語の以下のような集まりであるとする．

(1) すべての $i \neq j$ に対し，$L_i \cap L_j = \varnothing$，すなわち，二つ以上の言語に含まれる列は存在しない．

(2) $L_1 \cup L_2 \cup \ldots \cup L_k = \Sigma^*$，すなわち，すべての列はどれかの言語に属する．

(3) 各言語 L_i $(i = 1, 2, \ldots, k)$ は帰納的可算である．

このとき各 L_i は帰納的であることを示せ．

♣*! **問 9.2.5** L は RE 言語，$\overline{L}$ は非 RE であるとする．言語

$$L' = \{0w \mid w \in L\} \cup \{1w \mid w \notin L\}$$

を考える．L' や L' の補集合が，帰納的か，RE か，RE でないか，を答え，それを証明せよ．

♣ ! **問 9.2.6** 帰納的言語や RE 言語についての閉包性については，9.2.2 項の補集合演算以外は扱ってこなかった．帰納的言語や RE 言語が次の演算で閉じているか否か答えよ．閉じていることの証明には，形式的でなくても明確な構成法を与えよ．

*a) 集合和．

b) 共通部分．

c) 連接．

d) クリーネ閉包．

*e) 準同形写像．

f) 逆準同形写像．

9.3 テューリング機械の決定不能問題

決定可能性や帰納的可算性における位置づけがはっきりしている言語 L_d と L_u を用いて，他の決定不能問題や非 RE 問題を調べよう．証明には，還元を利用する．最初に扱う決定不能問題はすべてテューリング機械に関わるものである．事実本節の議論は，TM が受理する言語に関わる自明でない性質はすべて決定不能であるという「ライス (H. G. Rice) の定理」の証明で頂点に達する．9.4 節ではテューリング機械とその言語に関係しない決定不能問題を調べる．

9.3.1 還　　元

還元の概念は，8.1.3 項で紹介した．一般に，問題 P_1 の実例を問題 P_2 の同じ答の実例に変換するアルゴリズムは P_1 は P_2 に**還元する** (reduce) という．これは P_2 の難しさが P_1 以上であることの証明に使われる．したがって，P_1 が帰納的でなければ P_2 も帰納的でなく，P_1 が非 RE ならば P_2 も非 RE である．8.1.3 項で注意したように，難しさが既知の問題を，未知の難しさがそれ以上であることを知りたい問題に還元しなければならず，その逆ではいけない．

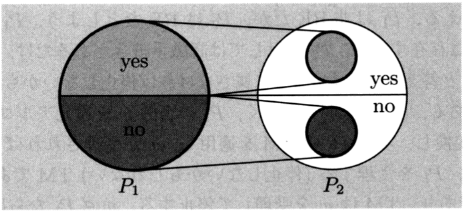


図 9.7　還元は肯定例を肯定例に，否定例を否定例に変える

図 9.7 に示したように，還元は，P_1 の yes の答を持つすべての実例を P_2 の yes の答を持つ実例に変え，P_1 の no の答を持つすべての実例を P_2 の no の答を持つ実例に変えなければならない．ここで，P_2 の実例がすべて P_1 の実例の変換先になっているか否かは本質的でない．そして実際，P_2 の実例のごく一部が P_1 の実例の変換先になっていることの方が普通である．

P_1 から P_2 への還元は，形式的には TM でそのテープ上に P_1 の実例を置いて開始し，P_2 の実例をテープ上に書き出して停止する TM をいう．実際上は，還元を一般的に，P_1 の実例を入力として受け取り，P_2 の実例を出力するコン

ピュータプログラムであるかのように記述する．テューリング機械とコンピュータプログラムの同値性から，還元をどちらの道具で記述してもよい．還元の重要性は，数多くの応用を持つ次の定理によって示される．

定理 9.7 P_1 が P_2 に還元できるとき，

a) P_1 が決定不能ならば，P_2 も決定不能である．

b) P_1 が非 RE ならば，P_2 も非 RE である．

証明 まず，P_1 が決定不能だとしよう．もし，P_2 が決定可能ならば，P_1 から P_2 への還元と P_2 の決定アルゴリズムをつないで，P_1 の決定アルゴリズムを構成できる．アイデアはすでに図 8.7 に示したが，より詳しく述べるために，P_1 の実例 w が与えられたとしよう．w に還元アルゴリズムを適用して，P_2 の実例 x に変換し，x に P_2 の決定アルゴリズムを適用する．その答が yes なら $x \in P_2$ である．P_1 は P_2 に還元されているから，P_1 における w の答も yes で，$w \in P_1$ である．同様に，$x \notin P_2$ ならば，$w \notin P_1$ である．結局,「$x \in P_2$ か」に対する答が何であってもそれはそのまま「$w \in P_1$ か」に対する正しい答である．

これは P_1 が決定不能だという仮定に反する．したがって，P_1 が決定不能ならば，P_2 も決定不能である．

次に (b) を考える．P_1 は非 RE だが，P_2 は RE だとしよう．P_1 から P_2 への還元アルゴリズムは存在するが，P_2 に対しては認識手続きがあるだけ，すなわち入力が P_2 に属せば yes と答えるが入力が P_2 に属さなければ停止しないかもしれない TM が存在するだけである．(a) の証明と同じく，P_1 の実例 w を還元アルゴリズムによって P_2 の実例 x に変換し，次に P_2 の TM を適用し，x が受理されれば w を受理する．

この手続きは，P_1 を受理する (停止しないかもしれない) TM である．$w \in P_1$ ならば，$x \in P_2$ であり，TM は w を受理して停止する．$w \notin P_1$ ならば，$x \notin P_2$ であり，TM は停止するかどうかわからないが，受理はしない．P_1 を受理する TM は存在しないと仮定しているから，これは矛盾であり，P_2 を受理する TM は存在しない．すなわち，P_1 が非 RE ならば P_2 も非 RE である． ▨

9.3.2 空集合を受理するテューリング機械

テューリング機械に関わる還元の例として，$\{0, 1\}$ 上の二つの言語 L_e と L_{ne} を調べよう．2 進列 w_i は，9.1.2 項の符号化により，TM M_i を表す．$L(M_i) = \varnothing$，すなわち，M_i がどの入力も受理しないならば，$w_i \in L_e$ である．したがって，L_e は，受理言語が空集合であるような TM の符号全体の集合である．一方，$L(M_i) \neq \varnothing$ ならば，$w_i \in L_{ne}$ である．すなわち，L_{ne} は少なくとも一つの列

を受理するような TM の符号全体の集合である．以下では，列 (符号) をそれが表す TM と同一視すると便利である．したがって，今述べた二つの言語は次のように定義できる．

- $L_e = \{M \mid L(M) = \varnothing\}$
- $L_{ne} = \{M \mid L(M) \neq \varnothing\}$

L_e と L_{ne} はともに $\{0,1\}$ 上の言語で，互いに補集合になっている．以下で，L_{ne} の方が "やさしい" ことを示そう．すなわち，L_{ne} は RE であるが帰納的でなく，一方，L_e は RE でない．

定理 9.8 L_{ne} は帰納的可算である．

証明 L_{ne} を受理する TM を示せばよい．図 9.8 に設計図を示した非決定性テューリング機械 (NTM) で説明するのが一番楽だろう．定理 8.11 より，M は決定性 TM に変換できる．

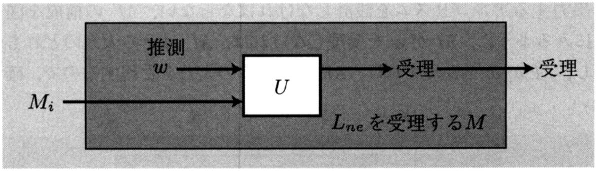


図 9.8 L_{ne} を受理する NTM

M は以下のように動作する．

(1) M は入力として M_i の符号を読む．

(2) 非決定性を利用して，M は M_i が受理するであろう入力 w を推測する．

(3) M は M_i が w を受理するか否か調べる．この部分では，M は L_u を受理する万能 TM U を模倣する．

(4) M_i が w を受理すれば，M はその入力 M_i を受理する．

こうして，M_i が一つでも入力列を受理すれば，M はその列を (もちろん他のどれをも) 推測でき，M_i を受理する．しかし $L(M_i) = \varnothing$ ならば，どのような w を推測しても M_i の受理状態には到達できず，M は M_i を受理しない．したがって，$L(M) = L_{ne}$ である． ▨

次の目標は，L_{ne} が帰納的でないことを証明することである．そのために L_u を L_{ne} に還元する．すなわち，入力 (M, w) に対し，次の条件を満たす TM M'

の符号を出力するアルゴリズムを記述する．ここで出力される M' は，M が w を受理するときはすべての入力列を受理し，M が w を受理しなければどの入力列も受理しない．すなわち，M が w を受理する必要十分条件は，M' が少なくとも一つの列を受理することである．その仕掛けは，自身の入力を無視して w 上の M の動作を模倣する M' を構成する点にある．M が w を受理すれば，M' は自身の入力を (それが何であっても) 受理する．したがって，$w \in L(M)$ は $L(M') \neq \varnothing$ と同値である．こうして，もし L_{ne} が帰納的だったら，M' を構成して $L(M') = \varnothing$ か否かを調べるという，M が w を受理するか否かを判定するアルゴリズムが得られる．

定理 9.9　L_{ne} は帰納的ではない．

証明　上で解説した方針にしたがって証明する．2 進符号化された対 (M, w) を入力として受け取り，M が w を受理するときそのときに限り $L(M') \neq \varnothing$ となる TM M' の符号を出力するアルゴリズムを設計しなければならない．M' の構成を図 9.9 に示した．後にみるように，M が w を受理しなければ，M' はその入力のどれも受理しない．しかし，M が w を受理すれば，M' はすべての入力を受理するので，確かに空集合ではない．

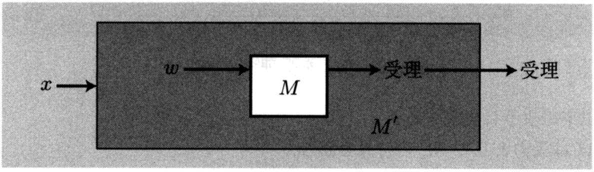


図 9.9　定理 9.9 で (M, w) から構成される TM M' の設計図．M' が任意の入力を受理する必要十分条件は M が w を受理することである．

M' は次の動作を行う．

(1)　M' は自身の入力を無視する．代わりに，入力テープ上の列を，M とその入力 w を表す列に置き換える．M' はある特定の対 (M, w) を表す列に対して設計されるので，次のような状態の列 $q_0, q_1, \ldots, q_n$ を持つようにできる (n は (M, w) を表す列の長さ)．ここで，q_0 は開始状態で，

　(a)　状態 q_i $(i = 0, 1, \ldots, n-1)$ では，2 進符号 (M, w) の $(i+1)$ 番目のビットを書き，1 マス右へ動いて状態 q_{i+1} に移る．

　(b)　状態 q_n では，M' は右へ動いて行って，出会った非空白記号をすべて

空白記号に置き換える (これは M' の入力 x が長さ n 以上のときに生じる).

(2) 状態 q_n で空白に達したら，そのヘッドをテープの (非空白部の) 左端に戻す.

(3) M' は万能 TM U を模倣する.

(4) U が受理状態に入れば，M' も受理状態に入る．U が受理状態に入ることがなければ，M' も受理しない.

上の M' の記述から，(M, w) の符号を M' の符号に変換するテューリング機械が設計できることを読者に十分納得してもらえるだろう．すなわち，L_u を L_{ne} に還元するアルゴリズムが存在する．M が w を受理すれば，M' はもともとどのような x がテープ上に置かれていようとそれを受理する．M' はもとの入力 x を無視するという言い方は適切でないかも知れない．TM における受理の定義によれば，動作を開始する前にテープに置かれている列が，TM が受理する列である．したがって，M が w を受理すれば，M' の符号は L_{ne} に属する.

逆に，M が w を受理しなければ，M' はその入力が何であったとしても，それを受理しない．それゆえ，M' の符号は L_{ne} に属さない．結局，(M, w) から M' を構成するアルゴリズムによって，L_u を L_{ne} にうまく還元することができたので，L_u が帰納的でないことから L_{ne} も帰納的でないといえる．定理の証明には，還元の存在で十分である．しかし，還元論法の威力を説明するために，さらに最後のステップを進めよう．L_{ne} が帰納的だったとすると，L_u の決定アルゴリズムを次のように構成できる.

(1) (M, w) を上の TM M' に変換する.

(2) 仮定された L_{ne} のアルゴリズムを適用して $L(M') = \varnothing$ か否かを決定する．$L(M') = \varnothing$ ならば M は w を受理せず，$L(M') \neq \varnothing$ ならば，M は w を受理する.

定理 9.6 より，L_u の決定アルゴリズムは存在しないことがわかっているから，L_{ne} が帰納的であるという仮定から矛盾が導かれ，L_{ne} は帰納的でない. ▨

これで，L_e の状況も判明した．L_e が RE だったら，定理 9.4 より L_e と L_{ne} はともに帰納的なはずである．L_{ne} は定理 9.9 から帰納的でないので，

定理 9.10　L_e は RE でない. ▨

9.3.3　ライスの定理と RE 言語の性質

L_e や L_{ne} といった言語が決定不能であるという事実は，RE 言語に関するすべての自明でない性質は決定不能であるという，ずっと一般的な事実の特殊な場合にすぎない．すなわち RE の自明でない性質は，言語がその性質を持つか否かをそれを受理する TM の符号から判定する (常に停止する) TM が存在し

ないという意味で，決定不能である．たとえば「文脈自由言語か」という性質を考えれば，与えられたTMが文脈自由言語を受理するか否かは，RE言語の自明でない性質はすべて決定不能であるという一般的な結果の特殊な場合として，決定不能である．

RE言語の**性質** (property) とは，単にRE言語のある集合である．したがって，文脈自由言語であるという性質はCFL全体の集合であり，空集合であるという性質は単純に空集合だけから成る集合 $\{\varnothing\}$ である．

RE言語の性質が**自明** (trivial) であるというのは，それが空である (すなわち，どのRE言語もそれを満たさない) かまたはRE言語全体であるときである．そうでないとき，性質は**自明でない** (nontrivial) という．

- 空の性質 $\varnothing$ は，空であるという性質 $\{\varnothing\}$ と異なることに注意せよ．

問題とその否定問題が異なる理由

直観的にはある問題とその否定とは同じ問題で，一つが解ければそれを使ってもう一方が解ける．最後のステップで，yesとnoを交換して，出力を反対にすればよい．この直観は，問題 (とその否定) が帰納的である限りでは，まったく正しい．

しかし，9.2.2項で議論したように，他に二つの可能性がある．まず，問題もその否定もともにREでさえない場合，どちらもTMで解けないという意味では同じといえる．しかし，面白いのは，L_u や L_{ne} に典型的にみられるように，一方がREで他方がREでない場合である．RE言語に対しては，入力 w がその言語に属する理由を探索するTMを設計できる．たとえば L_{ne} の場合，入力として与えられていたTM M に対して，M が受理する入力 w を探しそれが一つでも見つかれば M を受理するTMを設計した．このTMでは，M が空集合を受理するとき，M が L_{ne} に属さないことを知ることは決してできず，ただいつまでも M が受理されないというだけである．そしてそれは M に対する正しい振る舞いである．

一方，否定問題 L_e はREでないので，それに属す実例をすべて受理する方策はない．空集合を受理するTM M が与えられたとしよう．TM M の入力を順に調べていくことはできるが，いつまでたっても M が受理する列は見つけられず，しかもまだ調べていない列の中に M が受理する列がないことを確かめることもできない．したがって M は，たとえ L_e に属していても，決して受理されることはない．

言語の集合を言語として直接認識することはできない．言語は典型的には無限集合であり，TM の入力となる有限列として直接書き下すことはできない．しかし，TM の符号は受理言語が無限集合であっても有限の長さであるから，むしろ，それらの言語を受理する TM を認識 (受理) すべきである．したがって，$\mathcal{P}$ が RE 言語の性質のとき，言語 $L_{\mathcal{P}}$ は $L(M_i) \in \mathcal{P}$ を満たすテューリング機械 M_i の符号全体の集合であるとする．そして，性質 $\mathcal{P}$ の決定可能性を問題にするときには，言語 $L_{\mathcal{P}}$ の決定可能性を問題にする．

定理 9.11 (**ライスの定理**) RE 言語の自明でない性質は決定不能である．

証明　$\mathcal{P}$ を RE 言語の自明でない性質とする．最初に空言語 $\varnothing$ は $\mathcal{P}$ に属さないと仮定する (反対の場合は後で扱う)．$\mathcal{P}$ は自明でないので，$\mathcal{P}$ に属す空でない言語 L がある．TM M_L が L を受理するとしよう．

L_u を $L_{\mathcal{P}}$ に還元しよう．そうすれば L_u は決定不能なので $L_{\mathcal{P}}$ も決定不能である．還元を実現するアルゴリズムは対 (M, w) を入力とし，次の TM M' を出力する．M' の設計は図 9.10 に示した通りで，M が w を受理しなければ $L(M') = \varnothing$ であり，M が w を受理すれば $L(M') = L$ である．

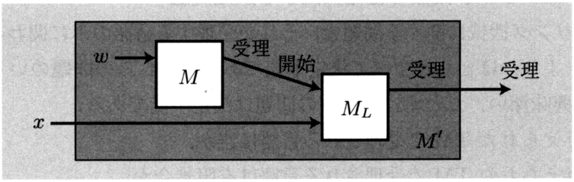


図 9.10　ライスの定理の証明に使う M' の構成

M' は 2-テープ TM である．1 本は w 上で M を模倣するために使う．還元を実現するアルゴリズムには M と w が入力として与えられ，M' の遷移を構成するために使われる．したがって，w 上での M の模倣は，M' に "組み込まれ" ており，M の遷移関数を自身のテープ上で読み込む必要はない．

M' のもう一つのテープは，M が w を受理したときに，入力 x 上で M_L を模倣するために使われる．再び，還元アルゴリズムは M_L の遷移関数を知っており，M' の遷移関数に "組み込まれる"．TM M' は次の動作を行うように構成される．

(1)　M を w 上で模倣する．w は M' の入力ではない．むしろ，定理 9.8 の証明で行ったように，M' は最初に M と w をそのテープ上に書き出し，その後，万能 TM U を模倣する．

(2) M が w を受理しなければ，M' はあと何もしない．M' は自身の入力 x を決して受理することはなく，$L(M') = \varnothing$ である．$\varnothing$ は性質 $\mathcal{P}$ に属さないので，M' の符号は $L_{\mathcal{P}}$ に属さない．

(3) M が w を受理すれば，M' は自身の入力 x 上で M_L を模倣する．したがって，M' はまさに言語 L を受理する．L は $\mathcal{P}$ に属すので，M' の符号は $L_{\mathcal{P}}$ に属す．

M と w から M' を構成するアルゴリズムがあることは確かめておいてほしい．このアルゴリズムは (M, w) を，$(M, w) \in L_u$ と $M' \in L_{\mathcal{P}}$ が同値になる M' に変換するので，L_u から $L_{\mathcal{P}}$ への還元アルゴリズムである．したがって，性質 $\mathcal{P}$ は決定不能である．

まだ終わっていない．$\varnothing$ が $\mathcal{P}$ に属す場合も考えなければならない．この場合は $\mathcal{P}$ を否定した性質 $\overline{\mathcal{P}}$，すなわち性質 $\mathcal{P}$ を持たない RE 言語の集合を考える．これまでの議論により，$\overline{\mathcal{P}}$ は決定不能である．すべての 2 進列はなんらかの TM を表すから，$\mathcal{P}$ に属す言語を受理しない TM の (符号の) 集合は $\overline{L_{\mathcal{P}}}$ であり，$\overline{\mathcal{P}}$ に属す言語を受理する TM の集合 $L_{\overline{\mathcal{P}}}$ に等しい．$L_{\mathcal{P}}$ が決定可能だったとしよう．すると帰納的言語の補集合は帰納的 (定理 9.3) なので，$L_{\overline{\mathcal{P}}}$ も決定可能になってしまい，矛盾である．よって $\varnothing \in \mathcal{P}$ の場合も $\mathcal{P}$ は決定不能である．▨

9.3.4 テューリング機械の仕様に関する決定問題

テューリング機械に関する問題で，それが受理する言語のみに関わる問題は，定理 9.11 より，ほとんどすべて決定不能である．これらの問題のいくつかは，それ自身興味深い．たとえば，以下の問題は決定不能である．

(1) 与えられた TM で受理される言語は空か．

(2) 与えられた TM で受理される言語は有限集合か．

(3) 与えられた TM で受理される言語は正則言語か．

(4) 与えられた TM で受理される言語は文脈自由言語か．

しかし，ライスの定理は TM に関するすべての問題が決定不能であることは意味しない．たとえば，TM が受理する言語でなく，TM の状態に関する問題なら，決定可能なこともある．

例 9.12 与えられた TM が 5 個の状態を持つか否かは決定可能である．この問題を解くアルゴリズムは単にその TM の符号を調べて，その遷移関数中に現れる状態の個数を数えればよい．

別の例として，TM が少なくとも五つの動作を行う入力があるか否かは決定可能である．TM が五つの動作を行う間に見ることのできるマスは，最初のヘッ

ド位置から左右4マスずつの9マス以下であることを思い起こせば，決定アルゴリズムは簡単だろう．したがって長さ5以下のすべての入力記号列上で，高々5回の動作を模倣することができる．この中に停止状況に達しないものがあれば，TMは少なくとも一つの入力上で5回の動作を行うと結論できる．▨

9.3.5　9.3節の演習問題

♣ ***問 9.3.1**　(他の列に加えて) 回文をすべて受理するテューリング機械の符号全体から成る集合は決定不能であることを示せ．

♣ **問 9.3.2**　大計算機株式会社は市場シェアの下落に対抗して，BWTMという高級テューリング機械の生産を決定した．BWTMは通常のテューリング機械と基本的には同じであるが，状態に2種類のラベル"ベル状態"と"笛状態"がついていてる．BWTMは新しい状態に移るとき，その状態のラベルに応じてベルを鳴らすか笛を吹くかする．与えられたBWTM M とその入力 w に対して，M が笛を吹くことがあるか否かという問題は決定不能であることを示せ．

♣ **問 9.3.3**　空白のテープから出発して，テープ上のどこかに1を書き出すようなTM M の符号から成る言語は決定不能であることを示せ．

♣ **!問 9.3.4**　ライスの定理により，以下のどの問題も決定不能である．それらはREか否か示せ．

a) $L(M)$ は少なくとも二つの語を含むか．

b) $L(M)$ は無限集合か．

c) $L(M)$ は文脈自由言語か．

*d) $L(M) = (L(M))^R$ か．

♣ **!問 9.3.5**　L は，二つのTMの符号と整数とから成る三つ組 (M_1, M_2, k) で，言語 $L(M_1) \cap L(M_2)$ が k 個以上の列を含むもの全体であるとする．L はREであるが帰納的ではないことを示せ．

♣ **問 9.3.6**　以下の問題は決定可能であることを示せ．

*a) 空白テープから出発していつかは非空白記号を書き出すようなTMの符号の集合．**ヒント**：TM M が m 個の状態を持っているとき，それが行う最初の m 回の動作に着目すればよい．

!b) 左へのヘッド移動を決して行わないTMの符号の集合．

!c) TM M をその入力 w 上で走らせたとき，どのマスも2回以上訪れないような対 (M, w) の集合．

♣ **!問 9.3.7**　以下の問題は帰納的可算でないことを示せ．

*a) TM M が入力 w 上で停止しないような対 (M, w) の集合．

b) $L(M_1) \cap L(M_2) = \varnothing$ と成るTMの対 (M_1, M_2) の集合．

c) $L(M_1) = L(M_2)L(M_3)$ となる三つ組 (M_1, M_2, M_3) の集合．すなわち，M_1 の受理言語は M_2，M_3 の受理言語の連接になっているような TM の三つ組の集合．

♣!! **問 9.3.8** 以下の集合がそれぞれ帰納的か，RE であるが帰納的でないか，RE でないか，を答えよ．

*a) すべての入力上で停止する TM の符合の集合．

b) どの入力上でも停止しない TM の符合の集合．

c) 少なくとも一つの入力上で停止する TM の符合の集合．

*d) 少なくとも一つの入力上で停止しない TM の符合の集合．

9.4 ポストの対応問題

本節では，テューリング機械に関する決定不能問題を，“現実的な” 問題，すなわち，抽象的なテューリング機械とはまったく関係のない普通の問題に還元する．まず，「ポスト (Post) の対応問題」(**PCP**) と呼ばれる問題から始めよう．これも抽象的ではあるが，テューリング機械ではなく列が関係する．目標は PCP の決定不能性を示すことと，PCP を他の問題に還元してそれらの決定不能性を示すことである．

PCP の決定不能性を証明するために，L_u を PCP に還元する．証明を容易にするために，“変形”PCP を導入して変形 PCP をもとの PCP に還元し，次に L_u を変形 PCP に還元する．この還元の連鎖を図 9.11 に示した．連鎖の最初の L_u の決定不能性はわかっているので，PCP も決定不能である．

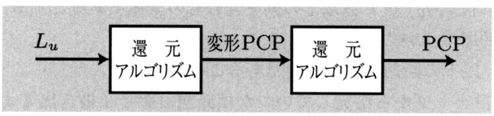


図 9.11 ポストの対応問題の決定不能性を証明する還元の連鎖

9.4.1 ポストの対応問題の定義

ポストの対応問題 (PCP, Post's Correspondence Problem) の実例は，あるアルファベット Σ 上の列の，同数の要素から成る二つのリストである．一般的にそれらをリスト A，B と呼び，$A = w_1, w_2, \ldots, w_k$，$B = x_1, x_2, \ldots, x_k$ と書く．各 i に対して対 (w_i, x_i) を**対応対** (corresponding pair) という．

PCP のこの実例が**解を持つ** (has a solution) というのは，1 個以上の整数の並び $i_1, i_2, \ldots, i_m$ が存在して，$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$ となる．すなわち，整数をリスト A，B 中の列の添え字としてみたときそれらが A，B で同じ列を生成するときである．並び $i_1, i_2, \ldots, i_m$ をこの実例の**解** (solution) という．このとき，ポストの対応問題は次のように述べることができる．

- 与えられた PCP の実例に対し，それが解を持つか否か．

例 9.13 $\Sigma = \{0, 1\}$ とし，リスト A, B を図 9.12 に示したものとする．この場合，解を持つ．たとえば，$m = 4, i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$，すなわち，並び $2, 1, 1, 3$ が一つの解である．実際，リスト A, B 中の対応する列を順に連接すれば，$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$ となる．この解しかないというわけではなく，たとえば $2, 1, 1, 3, 2, 1, 1, 3$ も解である． ▨

	リスト A	リスト B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

図 9.12 PCP の実例

言語としての PCP

与えられた PCP の実例が解を持つか否かを決定する問題を議論しているので，この問題を言語として表現する必要がある．PCP は任意のアルファベットを持つ実例を許しているので，9.1.2 項で状態やテープ記号の任意の集合を持つチューリング機械を符号化したときと同様に，PCP 言語は実際には PCP の実例を符号化したある固定されたアルファベット上の言語である．たとえば，PCP の実例が 2^k 個までの記号から成るアルファベットを持てば，各記号にそれぞれ異なる k ビットの 2 進符号を割り当てることができる．

各 PCP は有限アルファベットを持つので，各実例に対して k を見つけることができる．次にすべての実例を，0 や 1 と列を区切る “コンマ” から成る 3-記号アルファベットで符号化する．実例の符号化では，まず k を 2 進数で表し，後にコンマを置く．次に，リストの対応対のそれぞれを，k ビットの 2 進符号で符号化した列をコンマで区切ったもので表す．

	リスト A	リスト B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

図 9.13 PCP の別の実例

例 9.14 解を持たない実例を示そう．再び $\Sigma = \{0, 1\}$ とし，図 9.13 で与えた二つのリストを持つものとする．

図 9.13 の実例が解 $i_1, i_2, \ldots, i_m$ $(m \geq 1)$ を持つとする．$i_1 = 1$ である．実際 $i_1 = 2$ だとすると $w_2 = 011$ で始まる列と $x_2 = 11$ で始まる列とが等しくなるはずであるが，しかしこれらの列の最初の記号はそれぞれ 0 と 1 であるから，あり得ない．同様に，$w_3 = 101$ で始まる列が $x_3 = 011$ で始まる列と等しくなることはないので，$i_1 = 3$ でもない．

$i_1 = 1$ ならば，リスト A，B からの対応する列はそれぞれ，

$A : 10\ldots$

$B : 101\ldots$

で始まる．さて，i_2 はどうなるだろうか．

(1) $i_2 = 1$ のとき，$w_1w_1 = 1010$ で始まる列と $x_1x_1 = 101101$ で始まる列とは 4 番目の記号が異なるので，一致することはない．

(2) $i_2 = 2$ のとき，$w_1w_2 = 10011$ で始まる列と $x_1x_2 = 10111$ で始まる列とは 3 番目の記号が異なるので，一致することはない．

(3) よって $i_2 = 3$ だけが可能である．

$i_2 = 3$ のとき，整数の並び $1, 3, \ldots$ から作られる二つの列は

$A : 10101\ldots$

$B : 101011\ldots$

となる．ここまででは，これを延長しても解が得られないという直接的な証拠はない．しかし，解が得られないことを次のようにして証明できる．理由は，現在，$i_1 = 1$ としたときと同じ状況にあるという点にある．リスト B の列はリスト A の最後に 1 を追加した列である．したがって，不一致記号の出現を避けるためには $i_3 = i_4 = \ldots = 3$ でなければならない．したがって，リスト A の列がリスト B の列に追いつくことはできず，解は決して得られない．▨

PCP の部分解

例 9.14 では，可能な**部分解** (partial solution) は何かという，PCP の実例を解析する際によく使われる論法を使った．ここで部分解というのは，添え字の並び $i_1, i_2, \ldots, i_r$ で，$w_{i_1} w_{i_2} \ldots w_{i_r}$ と $x_{i_1} x_{i_2} \ldots x_{i_r}$ が等しくないかもしれないが，どちらか一方が他方の接頭列になっているものである．整数の並びが解であれば，その接頭部はすべて部分解である．したがって，部分解にどんなものがあるかを理解できれば，解にどんなものがあるのか議論できる．

しかし，PCP は (これから示すように) 決定不能であるから，すべての部分解を計算するアルゴリズムは存在しない．部分解は無限にあるかもしれず，さらに悪いことに，解につながる部分解が表す列 $w_{i_1} w_{i_2} \ldots w_{i_r}$ と $x_{i_1} x_{i_2} \ldots x_{i_r}$ の長さがどのくらい離れることができるかの上限も存在しない．

9.4.2 変 形 PCP

L_u を PCP に還元するには，**変形されたポストの対応問題** (MPCP, Modified Post's Correspondence Problem) と呼ばれる中間的形式を導入するほうが楽である．MPCP では，解の最初の対はリスト A，B の最初の対でなければならないという条件が追加される．より形式的には，MPCP の実例は二つのリスト $A = w_1, w_2, \ldots, w_k$，$B = x_1, x_2, \ldots, x_k$ であり，その解は

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

となる 0 個以上の整数の並び $i_1, i_2, \ldots, i_m$ である．MPCP の解では頭の添え字 1 を省略しているが，対 (w_1, x_1) が二つの列の頭にこなければならない．さらに，PCP の解は 1 個以上の整数の並びであったが，MPCP では $w_1 = x_1$ のとき空の並びも解になる (もっともそのような例はあまりおもしろいものではなく，以下で用いられることはない)．

例 9.15 図 9.12 のリスト A，B は MPCP の実例とみることもできるけれども，MPCP の実例としては解を持たない．実際，部分解は 1 から始まるのでそれが表す列は

$$A: 1 \ldots$$
$$B: 111 \ldots$$

から始まることに注意しよう．部分解の 2 番目の整数は w_2 も w_3 も 10 で始まることから 2 や 3 にはなり得ない．したがって次も 1 であり，

$A: 11\ldots$

$B: 111111\ldots$

ができる．この議論を無限に繰り返すことはできる．しかし，不一致をさけることができる添え字は1だけであり，B の列は常に A の列の3倍の長さを持ち，両者が一致することはない． ▨

PCP の決定不能性を示す重要なステップは，MPCP の PCP への還元である．後で，L_u を MPCP に還元して MPCP が決定不能であることを示すが，そうすれば PCP もまた決定不能性であることが証明されたことになる．実際，PCP がもし決定可能なら，MPCP も決定可能になり，したがって L_u も決定可能になってしまう．

MPCP のアルファベット Σ 上の実例が与えられたとき，PCP の実例を次のように構成する．最初に，新しい記号 $*$ を導入し，MPCP の実例中の各記号の間に $*$ を挟んで，PCP の実例を作る．しかし，A リストでは $*$ は Σ の記号の後に入れ，B リストでは Σ の記号の前に入れる．例外の一つは，上の手順で新しく作られた最初の対の第1項の頭に $*$ 記号を追加したもので，PCP の解がこの対から始まることを強制するために追加される．さらに PCP の最後に対 $(\$, *\$)$ を付け加える．この対は MPCP の解を模倣する PCP の解を閉じる役目をする．

さて上記の構成を厳密に記述しよう．リスト $A = w_1, w_2, \ldots, w_k$ と $B = x_1, x_2, \ldots, x_k$ を持つ MPCP の実例が与えられ，記号 $*$ と $\$$ はこの実例のアルファベット Σ には属さないものとする．PCP の実例 $C = y_0, y_1, \ldots, y_{k+1}$ と $B = z_0, z_1, \ldots, z_{k+1}$ を以下のように構成する．

(1) $i = 1, 2, \ldots, k$ に対し，y_i は w_i 中の各記号の後に $*$ を挟んだ列とし，z_i は x_i 中の各記号の前に $*$ を挟んだ列とする．

(2) $y_0 = *y_1$，$z_0 = z_1$ とする．すなわち，第0対は第1対と似ているがその第1項 y_1 の前に新たに $*$ が追加される．この PCP の実例では，同じ記号から始まる対は第0対だけであるから，PCP の解はすべて添え字0から始まらなければならない．

(3) $y_{k+1} = \$$，$z_{k+1} = *\$$．

例 9.16　図 9.12 を MPCP の実例と考える．上記の手順で構成される PCP の実例を図 9.14 に示す．　▨

	リスト C	リスト D
i	y_i	z_i
0	*1*	*1*1*1
1	1*	*1*1*1
2	1*0*1*1*1*	*1*0
3	1*0*	*0
4	$	*$

図 9.14　MPCP の実例からの PCP の実例の構成

定理 9.17　MPCP は PCP に還元される．

証明　証明の核心は上記の構成にある．最初に，$i_1, i_2, \ldots, i_m$ がリスト A,B を持つ MPCP の解だとしよう．したがって $w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$ である．ここで w_i を y_i で置き換え，x_i を z_i で置き換えると，ほとんど同じ列 $y_1 y_{i_1} y_{i_2} \ldots y_{i_m}$ と $z_1 z_{i_1} z_{i_2} \ldots z_{i_m}$ が得られる．違いは前者は頭の*が欠けていて，後者は最後の $*$ が欠けているだけである．すなわち

$$*y_1 y_{i_1} y_{i_2} \ldots y_{i_m} = z_1 z_{i_1} z_{i_2} \ldots z_{i_m} *$$

が成り立つ．ここで，$y_0 = *y_1$，$z_0 = z_1$ であるから，

$$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} = z_0 z_{i_1} z_{i_2} \ldots z_{i_m} *$$

である．最後の $*$ は添え字 $k+1$ を追加することで処理できる．$y_{k+1} = \$$，$z_{k+1} = *\$$ なので，

$$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k+1}$$

である．したがって $0, i_1, i_2, \ldots, i_m, k+1$ が構成された PCP の実例の解である．

逆の，構成された PCP の実例が解を持てばもとの MPCP も解を持つことも証明しなければならない．PCP のこの実例の解は，同じ記号で始まる列の対が第 0 対 (y_0, z_0) だけなので添え字 0 から始まり，同じ記号で終わるのは第 $(k+1)$ 対なので，添え字 $k+1$ で終わらなければならない．したがって，解は $0, i_1, i_2, \ldots, i_m, k+1$ と書ける．

このとき，$i_1, i_2, \ldots, i_m$ が MPCP の解であることが示せる．理由は，$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1}$ から記号 $*$ と \$ を除くと $w_1 w_{i_1} w_{i_2} \ldots w_{i_m}$ が得られる点にある．同様に，$z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k+1}$ から記号 $*$ と \$ を除くと，$x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$ が得られる．

$$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k+1}$$

であるから，

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

である．したがって，構成されたPCPが解を持てばもとのMPCPも解を持つ．

この定理の直前で述べた構成アルゴリズムは，解を持つMPCPの実例を解を持つPCPの実例に変換し，また，解を持たないMPCPの実例を解を持たないPCPの実例に変換する．こうしてMPCPはPCPに還元され，仮にPCPが決定可能であればMPCPも決定可能になる． ▨

9.4.3 PCPの決定不能性の証明の完成

本質的なアイデアは，MPCPの実例 (A, B) の部分解においてTM M の入力 w 上の計算を模倣することにある．すなわち，部分解は M のIDの列 $\#\alpha_1\#\alpha_2\#\alpha_3\#\ldots$ で，α_1 は M の初期IDでかつすべての i に対し $\alpha_i \vdash \alpha_{i+1}$ となるものになる．リスト B からの列はリスト A から作られる列より，M が受理状態に入らない限り，1ステップ(ID)先にすすんでいる．M が受理状態に入れば，リスト A の列がリスト B の列に追いついて，解を生じさせるような対が用意される．しかし，受理状態に入れなければ，これらの対が使われることはなく，解は存在しない．

MPCPの実例の構成を簡単にするために，定理8.12を使い，TMは決して空白を印刷せず，最初のヘッド位置から左へは，決して飛び出さないと仮定する．その場合，テューリング機械のIDは常に，空白でないテープ記号の列 α, β と状態 q を用いて $\alpha q \beta$ の形に書ける．ただし，ヘッドが α のすぐ右の空白マスを見ているときは，状態 q の右に空白記号を書くことはせず β は空列になる．したがって，列 $\alpha\beta$ はまさにテープの内容，入力とその右の以前ヘッドが訪れたことのあるマスの内容を表す．

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ を定理8.12を満たすTMとし，$w \in \Sigma^*$ を入力列とする．MPCPの実例を以下のように構成する．対応対を選んだ意図を理解するために，目標は M が受理IDに到達しない限りリスト A が作るID列をリスト B のID列より1ステップ遅らせることだったことを思い出そう．

(1) 最初の対は，

リスト A	リスト B
$\#$	$\#q_0w\#$

である．MPCPの規則よりすべての解がこの対から始まり，w 上での M

の模倣が開始される．

(2)　テープ記号と # は両リストから生成される列に自由に追加できる．対

リスト A	リスト B	
X	X	X は Γ の元
#	#	

を使って状態以外の記号を"コピー"できる．事実，これらの対を選択することによって，リスト A の列をリスト B の列に一致させながら延長すると同時に，リスト B の列の後に直前の ID の不変部分をコピーする．こうして M の動作列における次の ID をリスト B の最後に形成する準備をする．

(3)　M の動作を模倣するために，遷移規則を反映した対応対を加える．すべての $q \in Q - F$ (すなわち非受理状態 q) と $p \in Q$，$X, Y, Z \in \Gamma$ に対して，

リスト A	リスト B	
qX	Yp	$\delta(q, X) = (p, Y, R)$ のとき．
ZqX	pZY	$\delta(q, X) = (p, Y, L)$ のとき．Z は任意のテープ記号．
$q\#$	$Yp\#$	$\delta(q, B) = (p, Y, R)$ のとき．
$Zq\#$	$pZY\#$	$\delta(q, B) = (p, Y, L)$ のとき．Z は任意のテープ記号．

とする．これらの対は (2) の対応対と協力して A の列を B の列に一致させながら延長することによって，B 列の後に次の ID を付け加える役割を果たす．とくに，これらの対は現 ID から次の ID を作り出すのに必要な変更を決定するため，現在の状態とその隣のテープ記号を必要とし，新しい状態とテープ記号，ヘッドの移動は B 列の後に構成される ID に反映される．

(4)　B 列の最後にある ID が受理状態を含むときには，部分解が完全な解になれるようにする必要がある．そのために，M の本来の ID ではない "ID" を導入して，受理状態の TM がその両側にあるテープ上の記号をすべて消去するときの ID の変化を記述する．したがって，q が受理状態ならば，任意のテープ記号 X，Y に対し，対

リスト A	リスト B
XqY	q
Xq	q
qY	q

を追加する．

(5) 受理状態がテープ上の記号をすべて消去したら，最後には，B 列の最後の ID として，状態だけが単独で残る．したがって，二つの列の**残余** (remainder)(A 列に，B 列と一致させるために追加しなければならない B 列の接尾列) は $q\#$ である．それゆえ解を完成させるために，最後の対

リスト A	リスト B
$q\#\#$	$\#$

を使う．

以下では，上で作成した5種類の対応対をそれぞれ規則 (1) の対，規則 (2) の対，等々という．

例 9.18 TM $M = (\{q_1, q_2, q_3\},\ \{0,1\},\ \{0,1,B\},\ \delta,\ q_1,\ B,\ \{q_3\})$ で δ が

q_i	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	—	—	—

で与えられ，入力列が $w = 01$ であるとする．これを MPCP に変換しよう．簡単のために M は決して空白 B を書かないとしているので，ID に B が現れることはない．したがって，B を含むような対はすべて除くことができる．構成される対応対の全体をそれが作られる理由とともに図 9.15 に示した．

M は次の計算で入力 01 を受理する．

$$q_1 01 \vdash 1q_2 1 \vdash 10q_1 \vdash 1q_2 01 \vdash q_3 101$$

この計算を模倣する部分解の列はいずれ解に到達するが，まずは最初の対から始めなければならない．

$A : \#$

$B : \#q_1 01\#$

規則	リスト A	リスト B	理由
(1)	$\#$	$\#q_1 01\#$	
(2)	0	0	
	1	1	
	$\#$	$\#$	
(3)	$q_1 0$	$1q_2$	$\delta(q_1, 0) = (q_2, 1, R)$ より
	$0q_1 1$	$q_2 00$	$\delta(q_1, 1) = (q_2, 0, L)$ より
	$1q_1 1$	$q_2 10$	$\delta(q_1, 0) = (q_2, 0, L)$ より
	$0q_1\#$	$q_2 01\#$	$\delta(q_1, B) = (q_2, 1, L)$ より
	$1q_1\#$	$q_2 11\#$	$\delta(q_1, B) = (q_2, 1, L)$ より
	$0q_2 0$	$q_3 00\#$	$\delta(q_2, 0) = (q_3, 0, L)$ より
	$1q_2 0$	$q_3 10\#$	$\delta(q_2, 0) = (q_3, 0, L)$ より
	$q_2 1$	$0q_1$	$\delta(q_2, 1) = (q_1, 0, R)$ より
	$q_2\#$	$0q_2\#$	$\delta(q_2, B) = (q_2, 0, R)$ より
(4)	$0q_3 0$	q_3	
	$0q_3 1$	q_3	
	$1q_3 0$	q_3	
	$1q_3 1$	q_3	
	$0q_3$	q_3	
	$1q_3$	q_3	
	$0q_3 0$	q_3	
	$0q_3 1$	q_3	
(5)	$q_3\#\#$	$\#$	

図 9.15　例 9.18 の TM と入力から構成された MPCP の実例

この部分解を拡張する唯一の方法は，リスト A の列に残余 $q_1 01\#$ の接頭語を追加することだけである．したがって次に，規則 (3) から得られた動作模倣対の一つ $(q_1 0, 1q_2)$ を選ばなければならない．よって部分解は

$$A : \#q_1 0$$

$$B : \#q_1 01\#1q_2$$

となる．さらに規則 (2) のコピー対を使って部分解を次のように拡張する．

$$A : \#q_1 01\#1$$

$$B : \#q_1 01\#1q_2 1\#1$$

この時点で，規則 (3) の別の対を使って，M の動作を模倣できる．適切な対は

$(q_2 1, 0q_1)$ で，その結果

$$A : \#q_1 01\#1q_2 1$$

$$B : \#q_1 01\#1q_2 1\#10q_1$$

となる．次に規則 (2) を使えば三つの記号 #，1，0 を順に “コピー” できる．しかしそこまでコピーすることは，M が次にヘッドを左へ移動させるために状態の直前の記号 0 がその動作を表す規則 (3) の対に必要となることから，誤りとなる．したがって，# と 1 までをコピーし，部分解

$$A : \#q_1 01\#1q_2 1\#1$$

$$B : \#q_1 01\#1q_2 1\#10q_1\#1$$

を得る．適切な規則 (3) の対は $(0q_1\#, q_2 01\#)$ で，部分解は

$$A : \#q_1 01\#1q_2 1\#10q_1\#$$

$$B : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#$$

となる．次に再び規則 (3) の対 $(1q_2 0, q_3 10)$ を使い，受理状態に達する．

$$A : \#q_1 01\#1q_2 1\#10q_1\#1q_2 0$$

$$B : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#q_3 10$$

これ以降，規則 (2) の対を使って必要なら記号をコピーしながら，規則 (4) の対を使い ID から q_3 以外の記号を消していく．部分解の続きは

$$A : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#q_3 101\#q_3 01\#q_3 1\#$$

$$B : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#q_3 101\#q_3 01\#q_3 1\#q_3\#$$

となる．ID に残っているのは q_3 だけなので，最後に規則 (5) の対 $(q_3\#\#, \#)$ が適用でき，解が完成する．

$$A : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#q_3 101\#q_3 01\#q_3 1\#q_3\#\#$$

$$B : \#q_1 01\#1q_2 1\#10q_1\#1q_2 01\#q_3 101\#q_3 01\#q_3 1\#q_3\#\#$$ ▨

定理 9.19 ポストの対応問題は決定不能である．

証明 図 9.11 に示した還元の連鎖はすでにほとんど完成している．MPCP から PCP への還元は定理 9.17 で証明した．この節では L_u を MPCP へ還元する方法を示した．したがって，この還元が正しいこと，すなわち，

- M が w を受理する必要十分条件は構成された MPCP が解を持つことである

ことを示せば，定理の証明は完成する．

(必要性)　例 9.18 が基本的なアイデアを説明している．$w \in L(M)$ ならば，規則 (1) の対から始めて，w 上での M の計算を模倣できる．実際，規則 (3) の対を使って各 ID の状態を A の列にコピーすると同時に M の 1 回の動作を模倣し，規則 (2) の対を使ってテープ記号や # を必要なだけコピーする．M が受理状態に達したら，規則 (4) の対と最後に規則 (5) の対を使い，A 列を B 列に追いつかせて解が完成する．

(十分性)　構成された MPCP の実例が解を持てば，それは M が w を受理するためであることを示す必要がある．まず，MPCP なので任意の解は最初の対から始まらなければならず，次の部分解から始まる．

$$A : \#$$

$$B : \#q_0w$$

部分解が受理状態を含まない限り，規則 (4) や (5) の対は使えない．状態とその前後の記号を処理できるのは規則 (3) の対だけで，テープ上の他の記号や#は規則 (2) で扱われる．したがって，M が受理状態に達しない限り，部分解はすべて次の形式を持つ．

$$A : x$$

$$B : xy$$

ここで，x は入力 w 上での M の計算を表す ID の列に#とそれに続く ID α の接頭列をつなげたものである．残余 y は α の全体と次の#，α に続く ID の，α が x 中に含まれる部分に対応する部分，から成る．

特に，M が受理状態に到達しない限り，B 列は A 列より長く，部分解は解にならない．したがって，解があれば，M はある時点で受理状態に入る，すなわち，M は w を受理する．　▨

9.4.4　9.4 節の演習問題

♣　**問 9.4.1**　以下の PCP の実例が解を持つか否か答えよ．

*a)　$A = (01, 001, 10)$; $B = (011, 10, 00)$.

b)　$A = (01, 001, 10)$; $B = (011, 01, 00)$.

c)　$A = (ab, a, bc, c)$; $B = (bc, ab, ca, a)$.

♣ **!問 9.4.2**　PCP は決定不能であることを示したが，その際アルファベットに制限は加えなかった．PCP はそのアルファベットを $\{0, 1\}$ に制限したとしても決定不能であることを，一般の PCP を $\{0, 1\}$ 上の PCP に還元することによって示せ．

♣*! **問 9.4.3**　PCP のアルファベットを 1 記号集合 $\{0\}$ に制限しても，やはり決定不能か．

♣ **!問 9.4.4**　**ポストのタグシステム** (Post's tag system) はあるアルファベット Σ

上の列の対の有限集合と開始列から成る．(w, x) が対のとき，任意の列 y に対して，$wy \vdash yx$ と書く．すなわち，1回の動作で，“現在の” 列 wy から接頭列 w を取り去り，代わりに w と対をなす列 x を後に付け加える．TM のときと同様に，$\overset{*}{\vdash}$ を $\vdash$ の 0 回以上の繰り返しで定義する．与えれた対の集合 P と開始列 z に対し，$z \overset{*}{\vdash} \varepsilon$ か否かは決定不能であることを示せ．**ヒント**：TM M とその入力 w に対して，z を初期 ID に区切り記号# を付けた列とする．対の集合 P を，M の任意の ID がいずれはそれに続く ID に変換されるように構成せよ．さらに M が受理状態に達したら，現在の列がいずれはすべて消去されるように設計せよ．

9.5 他の決定不能問題

決定不能性が証明できる多くの問題を与えよう．基本的な技法は PCP を決定不能性を証明しようとする問題に還元することである．

9.5.1 プログラムに関する問題

通常のプログラミング言語であればどれでも，PCP の実例を入力としてその解をたとえば解の長さ (対の数) の順に，系統的に探すプログラムを書くことができる．その際，PCP は任意のアルファベットが許されるので，9.4.1 項の囲み記事「言語としての PCP」で議論したように，アルファベット中の記号を 2 進あるいは適当な固定アルファベットで符号化しなければならない．

上のプログラムに，それが解を見つけたらそのとき，停止するあるいは hello, world と印刷するなど，特定の動作をさせることができる．解が見つからないかぎり，プログラムはその動作をしない．したがって，プログラムが，停止する，hello, world と印刷する，特定の関数を呼び出す，ベルを鳴らす等々，ある種の自明でない動作をするか否かは決定不能である．事実，ライスの定理と同様の定理が成り立ち，プログラムの振る舞いに関する (すなわち，プログラム自体の文面や構文に関する性質以外の) 自明でない性質はすべて決定不能である．

9.5.2 CFG の曖昧さに関する決定不能性

プログラムはチューリング機械に似ているので，9.5.1 項の議論は驚くべきものではない．この節では，与えられた文脈自由文法はあいまいか否か，というコンピュータとはまったく無関係にみえる問題へ PCP を還元しよう．鍵となるアイデアは，PCP の実例の一つのリスト上で生成される列とその列を生成す

る添え字 (整数) 列を逆順に並べたものとをつなげて考えることである．このような列の集合は文脈自由文法で生成できる．もう一方のリストから作られる同様の列の集合も CFG で生成でき，したがってそれらの和集合も CFG で生成できる．このとき，もとの二つの文法の両方から生成される列が存在することと PCP が解を持つこととが同値になる．したがって，PCP が解を持つ必要十分条件は，和集合を生成する文法があいまいなことである．

アイデアをより正確に記述しよう．PCP の実例が $A = w_1, w_2, \ldots, w_k$, $B = x_1, x_2, \ldots, x_k$ から成るとする．リスト A に対して，変数として A だけを持つ CFG を構成する．終端記号は PCP の実例に使われたアルファベット Σ 中のすべての記号と，解を構成するための選ばれる対を示す**添え字記号** (index symbol) $a_1, a_2, \ldots, a_k$ から成る．ここで，記号 a_i はリスト A から w_i が選ばれリスト B から x_i が選ばれることを表す．リスト A に対する CFG は次の規則を持つ．

$$\begin{array}{lll} A & \rightarrow & w_1Aa_1 \mid w_2Aa_2 \mid \ldots \mid w_kAa_k \\ & & w_1a_1 \mid w_2a_2 \mid \ldots \mid w_ka_k \end{array}$$

この文法を G_A，G_A が生成する言語を L_A と書き，**リスト A の言語** (the language for list A) という．

文法 G_A で導出される終端記号列は $w_{i_1}w_{i_2}\ldots w_{i_m}a_{i_m}\ldots a_{i_2}a_{i_1}$ の形をしている．ここで，$i_1, i_2, \ldots, i_m$ $(m \geq 1)$ は 1 以上 k 以下の添え字である．G_A の文形式は，w_i の列と a_i の列の間に変数 A が一つだけ挟まった形をしており，導出の最後に右辺に A を含まない生成規則が適用される．したがって構文木は図 9.16 のようになる．

G_A はあいまいでなく，A から導出される終端記号列はすべてただ一つの導出を持つ．生成された記号列の後半の添え字列は各導出ステップでどの規則が使われるかを決定する．すなわち，与えられた添え字記号 a_i で終わる生成規則は $A \rightarrow w_iAa_i$ と $A \rightarrow w_ia_i$ の二つしかなく，前者は導出の途中で使われ，後者は導出の最後に使われる．

さて，与えられた PCP の実例のもう一方のリスト $B = x_1, x_2, \ldots, x_k$ を考えよう．このリストからもう一つの文法 G_B

$$\begin{array}{lll} B & \rightarrow & x_1Ba_1 \mid x_2Ba_2 \mid \ldots \mid x_kBa_k \\ & & x_1a_1 \mid x_2a_2 \mid \ldots \mid x_ka_k \end{array}$$

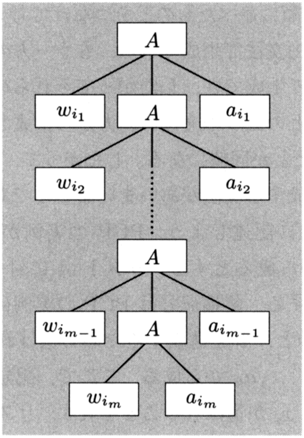


図 9.16 文法 G_A の構文木

が得られる．G_B が生成する言語を L_B と書く．G_A についての上の議論は G_B についても成り立ち，L_B に属す終端記号列は，その後半部分の添え字列が示すただ一つの導出を持つ．

最後に，二つのリストから構成された文法を組み合わせて，PCP の実例全体に対する文法 G_{AB} を構成する．G_{AB} は，

(1) 変数 A, B と開始記号，

(2) 規則 $S \to A \mid B$,

(3) G_A のすべての規則

(4) G_B のすべての規則

から成る．G_{AB} があいまいである必要十分条件は PCP の実例 (A, B) が解を持つことであることを示そう．それが次の定理の証明の核心となる．

定理 9.20 CFG があいまいであるか否かは決定不能である．

証明 PCP を CFG のあいまい性問題へ還元する方法はすでに与えた．PCP は決定不能であるから，この還元により，CFG のあいまい性問題は決定不能である．残っているのは還元が正しいこと，すなわち，以下を証明することである．

- G_{AB} があいまいである必要十分条件は PCP の実例 (A, B) が解を持つことで

ある．

(十分性) $i_1, i_2, \ldots, i_m$ が PCP の実例 (A, B) の解だとする．G_{AB} の二つの導出

$$S \Rightarrow A \Rightarrow w_{i_1} A a_{i_1} \Rightarrow w_{i_1} w_{i_2} A a_{i_2} a_{i_1} \Rightarrow \ldots$$
$$\Rightarrow w_{i_1} w_{i_2} \ldots w_{i_{m-1}} A a_{i_{m-1}} \ldots a_{i_2} a_{i_1} \Rightarrow w_{i_1} w_{i_2} \ldots w_{i_m} a_{i_m} \ldots a_{i_2} a_{i_1}$$
$$S \Rightarrow B \Rightarrow x_{i_1} B a_{i_1} \Rightarrow x_{i_1} x_{i_2} B a_{i_2} a_{i_1} \Rightarrow \ldots$$
$$\Rightarrow x_{i_1} x_{i_2} \ldots x_{i_{m-1}} B a_{i_{m-1}} \ldots a_{i_2} a_{i_1} \Rightarrow x_{i_1} x_{i_2} \ldots x_{i_m} a_{i_m} \ldots a_{i_2} a_{i_1}$$

を考えよう．$i_1, i_2, \ldots, i_m$ は解であるから，$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$ である．したがって，上の二つの導出は同じ終端記号列を生成する．明らかにこれらの導出は二つの異なる最左導出であり，G_{AB} はあいまいである．

(必要性) すでに G_A や G_B があいまいでないことはわかっている．したがって，G_{AB} の終端記号列が二つの導出を持つ可能性は，一つが $S \Rightarrow A$ の後 G_A の導出が続く場合で他方が $S \Rightarrow B$ から始まり G_B の導出が続く場合だけである．

これら二つの導出を持つ終端記号列の添え字列 $a_{i_m} \ldots a_{i_2} a_{i_1}$ は，等しい前半部 $w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$ を作り出すことから，PCP の解を表す． ▨

9.5.3 リスト言語の補集合

リスト A に対する L_A のような文脈自由言語があれば CFL の多くの問題の決定不能性が示せるが，その補集合 $\overline{L_A}$ からは CFL のもっと多くの決定不能性が得られる．$\overline{L_A}$ はアルファベット $\Sigma \cup \{a_1, a_2, \ldots, a_k\}$ 上の L_A に含まれない列から成る．ここで，Σ はある PCP の実例のアルファベットであり，$a_1, a_2, \ldots, a_k$ はその実例の対応対の添え字を表す記号である．

$\overline{L_A}$ の興味あるメンバーは，リスト A の列を並べた後にそれとは対応しない添え字列を続けた $\{w_1, w_2, \ldots, w_k\}^* \{a_1, a_2, \ldots, a_k\}^* - L_A$ の元である．しかし，$\overline{L_A}$ には単に $\Sigma^* \{a_1, a_2, \ldots, a_k\}^*$ に属さないものもある．

$\overline{L_A}$ は CFL である．L_A と異なり，$\overline{L_A}$ を生成する CFG を設計するのはやさしくないが，PDA，実際には決定性 PDA を構成することができる．

定理 9.21 リスト A に対する L_A の補集合 $\overline{L_A}$ は CFL である．

証明 リスト $A = w_1, w_2, \ldots, w_m$ 中の列のアルファベットを Σ とし，I をその添え字記号の集合 $I = \{a_1, a_2, \ldots, a_m\}$ とする．$\overline{L_A}$ を受理する決定性 PDA P は次のように動く．

(1) P は Σ を読んでいる間は，それをスタックにプッシュしていく．Σ^* の任意の列は $\overline{L_A}$ に属すので，その間 P は受理状態にある．

(2) P は I に属す添え字記号 a_i を読むと直ちに，スタックの記号を順にポップし

ていって，a_i に対応する列の逆順 w_i^R がでてくるか否かを検査する．

(a) 答が否ならば，これまでに読んだ入力とさらにそれに任意の列を付け加えた列はすべて $\overline{L_A}$ に属する．したがって，P は受理状態に入ったまま以降の入力を読んでいく．

(b) w_i^R がポップされてかつスタックの底記号が現れないならば，P は受理状態に入る．入力は P が受理しない L_A の元になっているかもしれないので，それがはっきりするまで，次の入力記号が I の元である限り (2) を繰り返す．

(c) w_i^R がポップされてかつスタックの底記号が現れたとき，ここまでの入力は L_A の列なので非受理状態に入る．しかし入力がさらに続けば，それは L_A の列ではないので，受理状態に入ったまま入力を読んでいく．

(3) I の記号を一つでも読んだ後で Σ の記号がきたら，入力は L_A の列の形になっていない．したがって，P は受理状態に入り，そのまま以降の入力を読んでいく． ▨

文脈自由言語の種々の決定不能性を示すのに，L_A や L_B，それらの補集合が様々に使われる．次の定理にそれらをまとめて示そう．

定理 9.22 G_1, G_2 を文脈自由文法，R を正則表現とする．以下の問題は決定不能である．

a) $L(G_1) \cap L(G_2) = \varnothing$ か．

b) $L(G_1) = L(G_2)$ か．

c) $L(G_1) = L(R)$ か．

d) あるアルファベット T に対し，$L(G_1) = T^*$ か．

e) $L(G_1) \subseteq L(G_2)$ か．

f) $L(R) \subseteq L(G_2)$ か．

証明 証明はそれぞれ PCP からの還元である．PCP の実例 (A, B) を，CFG と正則表現に関わる上記の問題のそれぞれに対し，答が yes になることと PCP が解を持つことが同値になるように変換する．実際には，PCP を定理で述べた問題に還元するだけでなく，場合によってはその否定問題に還元する．帰納的言語のクラスは補集合演算で閉じている (定理 9.3) から否定問題が決定不能ならもとの問題も決定不能である．

PCP の実例のアルファベットを Σ とし，添え字記号の集合を I とする．還元はすべて，L_A，L_B，$\overline{L_A}$，$\overline{L_B}$ がそれぞれ生成 CFG を持つことに基づく．これらの CFG は，9.5.2 項のように直接的に構成されるか，あるいはそれを受理する PDA から定理

6.14 によって変換される．

a) $L(G_1) = L_A$，$L(G_2) = L_B$ とする．このとき $L(G_1) \cap L(G_2)$ は本質的に PCP の実例の解の集合であり，共通部分が空であることと解が存在しないことが同値になる．技術的には，解を持つ PCP を共通部分が空でない生成言語を持つ二つの CFG に変換したので,「二つの CFG の生成言語の共通部分が空でないか」という問題が決定不能であることを証明したことになる．しかし，証明の導入部で触れたように，問題の否定が決定不能であることともとの問題が決定不能であることは同値である．

b) CFG は生成言語の和集合演算で閉じているから，$\overline{L_A} \cup \overline{L_B}$ を生成する CFG G_1 を構成できる．$(\Sigma \cup I)^*$ は正則言語であるから，それを生成する CFG G_2 を構成できる．$\overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$ であるから，$L(G_1)$ に含まれないのはもとの PCP の実例の解に対応する列だけであり，$L(G_2)$ には $(\Sigma \cup I)^*$ のすべての列が含まれる．したがって，これらが等しいのは PCP が解を持たないときそのときに限られる．

c) 証明は本質的に (b) と同じであるが，R を正則表現 $(\Sigma \cup I)^*$ とする．

d) $\overline{L_A} \cup \overline{L_B}$ と閉包 T^* が等しくなる可能性のあるアルファベット T は $\Sigma \cup I$ だけであるから，(c) の証明がそのまま適用できる．

e) G_1 は $(\Sigma \cup I)^*$ を生成する CFG，G_2 は $\overline{L_A} \cup \overline{L_B}$ を生成する CFG であるとする．このとき，$L(G_1) \subseteq L(G_2)$ である必要十分条件は $\overline{L_A} \cup \overline{L_B} = (\Sigma \cup I)^*$，すなわち，PCP の実例が解を持たないことである．

f) (e) の証明がそのまま成り立つが，R を正則表現 $(\Sigma \cup I)^*$ とし，$L(G_1) = \overline{L_A} \cup \overline{L_B}$ とする．▨

9.5.4　9.5 節の演習問題

♣ ***問 9.5.1**　L を，少なくとも一つの回文を含む言語を生成する文脈自由文法を表す符号の全体とする．L は決定不能であることを示せ．**ヒント**：PCP の実例に対して，それが解を持つときそのときに限り回文を含むような文法を構成せよ．

♣ **!問 9.5.2**　$\overline{L_A} \cup \overline{L_B}$ が正則言語である必要十分条件はそれが $(\Sigma \cup I)^*$ になること，すなわち，PCP の実例 (A, B) が解を持たないことであることを示せ．またこれによって，CFG が正則言語を生成するか否かは決定不能になることを示せ．**ヒント**：(A, B) が解を持つと仮定し，wx が $\overline{L_A} \cup \overline{L_B}$ に属さないとする．ここで，w は解に対応するこの実例のアルファベット上の列で，x は解を表す添え字記号の列の逆順である．準同形写像 $h(0) = w, h(1) = x$ と定義すると，$h^{-1}(\overline{L_A} \cup \overline{L_B})$ はどんな言語になるか．正則言語は逆準同形写像，補集合の演算で閉じいることと，正則言語の反復補題とを使って，$\overline{L_A} \cup \overline{L_B}$ が正則でないことを示せ．

♣ **!!問 9.5.3**　CFL の補集合がまた CFL であるか否かは決定不能である．問 9.5.2

を使って，CFL の補集合が正則か否かは決定不能であることは示せるが，それは別のことである．最初の主張を証明するには，PCP の実例 (A, B) の解でない列を表す別の言語を定義する必要がある．L_{AB} を以下の条件の (1), (2), (3) と (4)〜(7) の少くとも一つを満たす列 $w\#x\#y\#z$ の集合とする．

(1) w と x は PCP の実例のアルファベット上の列である．

(2) y と z はその添え字記号の集合 I 上の列である．

(3) $\#$は Σ にも I にも属さない記号である．

(4) $w \neq x^R$.

(5) $y \neq z^R$.

(6) x^R は添え字列 y にしたがってリスト B から作られる列ではない．

(7) w は添え字列 z^R にしたがってリスト A から作られる列ではない．

L_{AB} は，(A, B) が解を持たなければ，$\Sigma^*\#\Sigma^*\#I^*\#I^*$ に属す列全体から成るが，とにかく CFL である．$\overline{L_{AB}}$ が CFL である必要十分条件は (A, B) が解を持たないことであることを示せ．ヒント：問 9.5.2 の逆準同形技法と，問 7.2.5(b) のヒントにあるように部分列の長さを制限するためにオグデンの補題を使え．

9.6 第9章の要約

◆**帰納的言語と帰納的可算言語**：テューリング機械で受理される言語は帰納的可算 (RE) であるといい，すべての入力に対して停止する TM で受理される言語は帰納的であるという．

◆**帰納的言語と RE 言語の補集合**：帰納的言語のクラスは補集合演算で閉じており，言語とその補集合がともに RE ならば，その言語は帰納的である．したがって，RE であるが帰納的でない言語の補集合は RE ではない．

◆**決定可能性と決定不能性**：「決定可能」は「帰納的」の同義語であるが，通常，言語は帰納的であるといい，問題 (や問題を翻訳した言語) は決定可能であるという．言語が帰納的でなければ，それが表す問題は決定不能である．

◆**言語 L_d**：この言語は 0 や 1 の列で，それが表す TM がそれ自身を受理しないような列の全体である．L_d は RE でない言語，すなわちどの TM でも受理されない言語のよい例である．

◆**万能言語**：万能言語 L_u は，TM と解釈できる列の後に入力列が続く列で，TM がその入力を受理するもの全体から成る．L_u は RE であるが帰納的でない言語のよい例である．

◆**ライスの定理**：テューリング機械が受理する言語の自明でない性質は決定不能である．たとえば，受理言語が空であるようなテューリング機械を表す符合の集合はライスの定理により決定不能である．実際，その言語は RE でないが，その補集合，

少なくとも一つの列を受理する TM の符号の集合は RE でかつ帰納的でない．

◆**ポストの対応問題**：この問題は，同数の列から成る二つのリストが与えられたとき，一連の対応する列を並べて，同じ列を作ることができるか否かを問う．PCP は決定不能な問題の重要な例である．他の問題に還元してその決定不能性を示すのには PCP を選ぶとよい．

◆**決定不能な文脈自由言語問題**：PCP からの還元により，CFL や CFG に関する多くの問題の決定不能性が示せる．たとえば，ある CFL が他の CFL に含まれるか，二つの CFL の共通部分は空か，などは決定不能である．

9.7　第 9 章の参考文献

万能言語の決定不能性は本質的にテューリング [9] の仕事であるが，言語と受理状態による受理ではなく，算術関数の計算とその停止性を使って表された．ライスの定理は [8] による．

ポストの対応問題の決定不能性は，[7] で示されたが，本書の証明は R. W. Floyd による未公表のノートにある．ポストシステム (問 9.4.4) の決定不能性は [6] による．

文脈自由言語に関する問題の決定不能性についての基本的な論文は [1] と [5] である．しかし，CFG があいまいか否かが決定不能であるという事実は，Cantor[2], R. W. Floyd [4]，Chomsky と Schutzenberger[3] によって独立に発見された．

[1] Y. Bar-Hillel, M. Perles, and E. Shamir, "On formal properties of simple phrase-structure grammars," *Z. Phonetik. Sprachwiss. Kommunikationsforsch.* **14** (1961), pp.143–172.

[2] D. C. Cantor, "On the ambiguity problem in Backus systems," *J. ACM* **9**:4 (1962), pp.477-479.

[3] N. Chomsky and M. P. Schutzenberger, "The algebraic theory of context-free languages," *Computer Programming and Formal Systems* (1963), North Holland, Amsterdam, pp.118–161.

[4] R. W. Floyd, "On ambiguity in phrase structure languages," *Communications of the ACM* **5**:10 (1962), pp.526–534.

[5] S. Ginsburg and G. F. Rose, "Some recursively unsolvable problems in ALGOL-like languages," *J. ACM* **10**:1 (1963), pp.29–47.

[6] M. L. Minsky, "Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines," *Annals of Mathematics* **74**:3 (1961), pp.437–455.

[7] E. Post, "A variant of a recursively unsolvable problem," *Bulletin of the AMS* **52** (1946), pp.264–268.

[8] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Transactions of the AMS* **89** (1953), pp.25–59.

[9] A. M. Turing, "On computable numbers with an application to the Entscheidungsproblem," *Proc. London Math. Society* **2**:42 (1936), pp.230–265.

10 実行不能な問題

本章では，計算可能であるか否かの問題を，効率よく計算できるか否かという視点にまで引き下げて論じることにしよう．ここでは，考察の対象を可解な問題のみに限定する．そして，入力のサイズに関して多項式で表される程度の計算時間で実行を終えるテューリング機械によって計算可能であるのは，それらのうちのどの問題であるのかを問う．まず，8.6.3 項に記した二つの重要な事柄を復習しておこう．

- 通常の計算機を用いて多項式時間で解くことのできる問題の全体は，テューリング機械で多項式時間で解くことのできる問題の全体と一致する．
- 経験が示すところによれば，多項式時間で解くことのできる問題と指数時間またはそれ以上の時間を必要とする問題の間にある境界線は，極めて本質的なものである．実際に我々が直面する問題で多項式時間を要するものは，ほとんど例外なく，我々が待つことができる程度の時間内に解くことができる．それに対して，指数時間を要する問題は，ほとんどの場合，小さなサイズの入力に対してしか解くことができない．

本章で紹介するのは「実行不能性」の理論，すなわち，ある問題が多項式時間では解けないことを示すための手法に関する理論である．我々は，ある特定の問題から出発する—— それはブール式が**充足可能**かどうかを問う問題，すなわち，式に含まれる変数への真理値 TRUE および FALSE の適当な割り当てにより式の値を真にすることが可能かどうかを問う問題である．この問題は，決定不能な問題に対して L_u や PCP が演じたのと同様の役割を，実行不能な問題に対して演じる．もう少し明確にいうと，ブール式の充足可能性は多項式時間では決定可能でない（であろう）ことを示す「クックの定理」が我々の出発点である．次に，我々は，この問題を他の多くの問題に還元する方法を示し，それによって，それらの問題が同様に実行不能であることを導く．

ここでは，問題が多項式時間で解けるかどうかを考察の対象とするので，還元の概念についても変更の必要が生じる．一つの問題の入力例を別の問題の入力例に変換するアルゴリズムの存在を示すだけでは，もはや十分ではない．還元のアルゴリズム自身も高々多項式時間しか必要としないものでなければならない．そうでないと，たとえ，もとの問題が実行不能であるとしても，この還元から目標の問題の実行不能性を導くことはできない．そこで，我々は，10.1 節で「多項式時間還元」という概念を導入する．

決定不能性の理論において導かれた結論と，実行不能性の理論において導かれる結論の間には，質的な点で，もう一つ重要な違いがある．第 9 章で与えた決定不能の証明は，いわば，異論をはさむ余地のない種類のものであった．それらはテューリング機械の定義と普通の数学のみに依存する証明であった．ところが，それとは対照的に，我々がこれから述べようとする実行不能問題に関する結果はすべて，その成立が強く信じられてはいるもののまだ証明されてはいない一つの仮定，しばしば $\mathcal{P} \neq \mathcal{NP}$ と呼ばれる仮定，に基礎を置き，それから導かれるものである．

この仮定とは次のようなものである．多項式時間で動作する非決定性の TM によって解くことのできる問題のクラスには，多項式時間で動作する決定性の TM によっては（たとえ，どんなに高次の多項式を許すとしても）解くことのできないいくつかの問題が含まれる．このカテゴリーに属すように**思われる**問題は，文字通り，数千個もあげることができる．それらはいずれも多項式時間の NTM によって容易に解くことができるが，一方，多項式時間の DTM（あるいは，計算機のプログラムといっても同じことであるが）で解を求める方法は知られていない．さらに，実行不能性の理論から導かれる重要な結論の一つは，これらのすべての問題に対し多項式時間の決定性の解が存在するか，つまり，数世紀にわたり我々の追及の手を逃れてきた巧妙な解がすべての問題について存在するか，または，どの一つの問題も多項式時間の決定性の解を持たないか，すなわち，すべての問題が（広い意味での）指数時間を必要とするか，のいずれかが成り立つということである．

10.1 クラス $\mathcal{P}$ とクラス $\mathcal{NP}$

本節では，実行不能性の理論において用いられる基本的な概念を紹介する．取り上げる事柄は，決定性 TM によって多項式時間で解くことのできる問題全体のクラス $\mathcal{P}$ と非決定性 TM によって多項式時間で解くことのできる問題全体のクラス $\mathcal{NP}$，そして，多項式時間還元の手法である．さらに，我々は「NP

多項式と指数の中間に位置するものはあるか？

入門的な議論を行う際，さらにその後の議論でも，我々は，すべてのプログラムは多項式時間［ある整数 k に対し $O(n^k)$ の時間］で走るか，指数時間［ある定数 $c>0$ に対し $O(2^{cn})$ の時間］で走るか，それともさらにそれより大きな時間で走るかのいずれかであるかのように想定して話を進めることが多い．実際，通常扱われる問題に対する既知のアルゴリズムは，たいてい，多項式時間か指数時間のいずれかのカテゴリーに属する．しかし，多項式時間と指数時間の中間に位置する計算時間が存在することもまた紛れもない事実である．我々が指数時間というとき，我々は実際には「すべての多項式時間より大きい任意の時間」を想定している．

多項式と指数の間に位置する関数の一例は $n^{\log_2 n}$ である．この関数は n に関するすべての多項式よりも速く増大する．なぜならば，$\log_2 n$ は，どんな定数 k よりも，いつかは（すなわち，十分大きな n に対しては）大きくなるからである．一方，$n^{\log_2 n} = 2^{(\log_2 n)^2}$ が成り立つ．不審に思う方は両辺の対数をとってみていただきたい．この関数は，任意の $c>0$ に対し，2^{cn} よりゆっくり増大する．すなわち，正の定数 c がどんなに小さい値であるとしても，cn はいつかは $(\log_2 n)^2$ より大きくなる．

完全性」という概念の定義も与える．これは，$\mathcal{NP}$ に属すある種の問題に付随する性質であり，その問題が（多項式時間の差を無視するとき）$\mathcal{NP}$ に属すどの問題と比べてもそれと同程度かまたはそれより難しいことを意味する．

10.1.1 多項式時間で解ける問題

テューリング機械 M は，長さ n の任意の入力 w に対し，w を受理するか否かに拘わらず，高々$T(n)$ 回の動作ののちに必ず停止するとき，その**時間計算量**は $T(n)$ である［または，$T(n)$ の"計算時間"を持つ］という．この定義は，$T(n)=50n^2$ や $T(n)=3^n+5n^4$ などの任意の関数 $T(n)$ に対して適用される．ただし，我々が主に興味を持つのは，$T(n)$ が n に関する多項式である場合である．言語 L がクラス $\mathcal{P}$ に属すとは，ある多項式 $T(n)$ が存在して，時間計算量 $T(n)$ のある決定性 TM M に対し $L=L(M)$ が成り立つことである．

10.1.2 例：クルシュカルのアルゴリズム

読者は，効率のよい解を持つ問題をすでにたくさんご存じであろうと思う．そのような問題のいくつかは，「データ構造とアルゴリズム」に関する科目の中で

学習の題材として取り上げられたかもしれない．それらの問題はほとんどが $\mathcal{P}$ に属している．ここでは，そのような問題の中から一つを選んで考察することにしよう．我々が取り上げるのは，与えられたグラフに対し最小の重みを持つ極大木（minimum-weight spanning tree, MWST）を求めよ，という問題である．

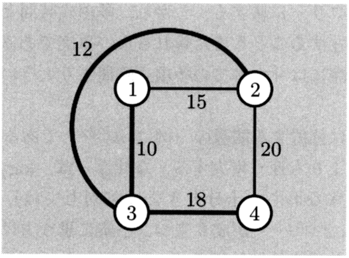


図 10.1　グラフの例；最小の重みを持つ極大木を太線で示す．

直観的にいうと，グラフとは図 10.1 に示す図形のようなものである．グラフには，頂点と，いくつかの頂点の対に対しその 2 頂点を結ぶ辺とがある．図 10.1 の例では各頂点に 1 から 4 までの番号が振られている．各辺は整数の重みを持つ．**極大木** (spanning tree) とは，辺を要素とする集合で，すべての頂点がそれらの辺を通して連結であり，かつ，閉路を含まないものである．極大木の例を図 10.1 に示す．太線で表示されている三つの辺の集合がそれである．**最小の重み** (minimum-weight) を持つ極大木とは，すべての極大木の中で辺の重みの和が最小のものである．

MWST を求めるための "欲張り (greedy)" アルゴリズムで，**クルシュカル (Kruskal) のアルゴリズム**[1]と呼ばれる，よく知られたアルゴリズムがある．このアルゴリズムの基本的なアイデアの概略は，次のようなものである．

(1)　各頂点に対し，それまでに選択された辺のいくつかを用いて構成される**連結成分** (connected component) で，その頂点を含むものを一つ対応させ，記憶する．アルゴリズムの実行開始の時点では，まだどの辺も選択さ

---

[1] J. B. Kruskal Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem, " *Proc. AMS* **7**:1 (1956), pp.48–50.

れていないので，すべての頂点に対し，その頂点のみから成る連結成分を対応させる．

(2) まだ取り上げられていない辺のうちで重みが最小の辺を一つ選ぶ．重みが最小の辺が複数個あるときは，それらの中からどれでも好きなものを選べばよい．選ばれた辺が，この時点では異なる連結成分に属する二つの頂点を結ぶ辺である場合：

  a) その辺を極大木の一部として選択する．

  b) さらに，その辺の二つの端点を含む二つの連結成分を併合して一つの連結成分にする．それには，一方の連結成分に含まれるすべての頂点の連結成分番号を，もう一方の連結成分番号と同じものに変更すればよい．

一方，選ばれた辺が同じ連結成分に属する二つの頂点を結ぶ辺である場合，この辺を極大木に含めることはしない．この辺を含めると閉路ができてしまうからである．

(3) 上記の辺を選ぶ操作を，すべての辺が選ばれ尽くすか，または，極大木の構成要素として選択された辺の個数が頂点の総数より一つ少ない個数になるまで続ける．後者の終了条件が満たされるのは，すべての頂点が一つの連結成分に属しているときであり，したがって，ここで操作を終了してよいことに注意されたい．

例 10.1 図 10.1 のグラフに対し，クルシュカルのアルゴリズムを適用してみよう．まず，辺 $(1,3)$ を選ぶ．この辺が最小の重み 10 を持つからである．頂点 1 と 3 は，最初，異なる連結成分に属しているので，我々はこの辺を選択する．そして，さらに，1 と 3 に同じ連結成分番号，たとえば，“連結成分 1” を割り振る．重みの順で次に小さな辺は，重み 12 を持つ辺 $(2,3)$ である．頂点 2 と 3 は異なる連結成分に属するので，我々はこの辺を選択する．さらに，頂点 2 を “連結成分 1” の中に併合する．3 番目に選ばれる辺は，重み 15 を持つ辺 $(1,2)$ である．しかし，1 と 2 はこの時点ですでに同じ連結成分に属している．そこで，我々はこの辺を選択することはせずに，次の辺に進む．4 番目の辺は $(3,4)$ である．頂点 4 は “連結成分 1” に含まれていないので，我々はこの辺を選択する．ここまで進んだところで，我々は頂点数が 4 のグラフの極大木とし

て三つの辺を得たことになる．これ以上進める必要はないので，ここで実行を終了する． ▨

このアルゴリズムを，（テューリング機械ではなく計算機の上で，）m 個の頂点と e 個の辺を持つグラフに対し $O(m + e \log e)$ 時間で実行するようなプログラムとして実現することが可能である．しかし，ここでは，それより単純でわかりやすい実現の方法を説明しよう．この方法は，ある一連の仕事を e 回繰り返す．各時点において，各頂点が属す連結成分を表の形で記憶しておく．まだ選ばれていない辺の中から最小の重みを持つ辺を選び出す．この作業に要する時間は $O(e)$ である．選ばれた辺の二つの端点の各々が属す連結成分を見つける．この作業は $O(m)$ 時間でできる．二つの端点が異なる連結成分に属す場合，該当する各頂点の連結成分番号を書き換えることにより両者を併合する．この作業は，頂点の表を上から下まで順に眺めていけばよいので，$O(m)$ 時間で行うことができる．したがって，このアルゴリズムの実行に要する時間の合計は $O(e(e+m))$ である．入力の「サイズ」として，仮に，e と m の和をとることにすれば，この実行時間は入力のサイズに関する多項式の時間である．

上に述べたアイデアをテューリング機械に置き換えようとすると，いくつかの問題点に直面することになる．

- アルゴリズムについて考える際，たとえば MWST を求める問題で解として辺の表を求めたように，形式は様々であるが，何らかの形の出力を要求する“問題”が考察の対象となることが多い．一方，通常のテューリング機械の理論では問題を言語としてのみ扱い，出力は yes または no のみ，あるいは，受理するか受理しないかのいずれかのみ，を考える．たとえば，MWST の問題は次のように解釈される．「グラフ G と限界値 W が与えられたとき，G は重みが W 以下の極大木を持つかどうかを判定せよ.」この問題では極大木を具体的に求めることは要求されていないので，この問題は，我々がすでにおなじみの MWST 問題よりもやさしい問題のように見えるかもしれない．しかし，実行可能性の理論においてそのことはあまり大きな障害にはならない．実行可能性の理論で我々が通常興味の対象とするのは，与えられた問題がやさしいことを示すことではなく，難しいことを示すことだからである．この観点から見るとき，ある問題を yes/no 問

題として解釈したものが難しい問題であるならば，そのことから，完全な解の計算を要求される標準的な解釈の問題もやはり難しい問題であることが導かれる，という事実こそが大事なのである．

- あまり厳密でない議論では，グラフの「サイズ」として頂点や辺の個数を考えることが多い．しかし，TM への入力は有限アルファベット上の文字列である．そのため，問題の構成要素が頂点や辺である場合にはそれらを適当に符号化する必要が生じる．この結果として，テューリング機械への入力は，直観的な入力の「サイズ」よりも若干長くなるのが普通である．しかし，この差は重要なものではない．その理由を 2 点述べる．
  (1)　TM への入力列によって与えられるサイズと直観的に問題を捉えたときの入力によって与えられるサイズとの差は，小さな因子を超えるものではなく，通常は入力サイズの対数倍程度でしかない．したがって，一方の尺度で測って多項式時間で実行できるならば，他方の尺度で測ってもやはり多項式時間で実行することができる．
  (2)　入力を表現する文字列の長さは，実際のコンピュータが入力を得るために読まなければならないバイト数の尺度として，直観的な入力のサイズよりもむしろ正確なものである．たとえば，一つの頂点を一つの整数によって表現しようとする．このとき，その整数を表現するのに必要なバイト数は整数の大きさの対数に比例するものである．厳密でない説明では，入力サイズを「1 頂点あたり 1 バイト」と想定してしまいがちだが，それは正しくない．

例 10.2　MWST 問題への入力であるグラフと重みの制限値に対し，符号（コード）化の一例を考えてみよう．コードは，0，1，左カッコ，右カッコ，コンマの 5 個の記号から成る．

(1)　各頂点に 1 から m までの整数を割り振る．

(2)　コードの先頭部分に m の値の 2 進表示と重みの制限値 W の 2 進表示をおく．両者の間にコンマをおき，両者を分離する．

(3)　頂点 i と頂点 j を結ぶ辺で重み w のものが存在するとき，(i,j,w) をコードの中におく．整数 i, j, w は 2 進数として符号化する．一つの辺の内部での i と j の順序や，コード内における辺の出現の順序は任意でよい．

図10.1のグラフを，重みの制限値を$W = 40$として，上の方法で符号化すると，次のようなコードが得られる．

$$100, 101000(1, 10, 1111)(1, 11, 1010)(10, 11, 1100)(10, 100, 10100)$$
$$(11, 100, 10010)$$ ▨

MWST問題への入力を例10.2に示した方法で表現するとき，長さnの入力列では高々$O(n/\log n)$個の辺が表現される．辺の個数が非常に少ない場合には，頂点の個数mがnに関して指数関数的になることもあり得る．しかし，辺の個数eが$m-1$以上でない場合，グラフは連結にはならず，したがって，そのようなグラフでは，辺の状況にかかわらず，MWSTは存在しない．このような考察により，頂点数が少なくとも$n/\log n$の何分の一か以上でない場合には，そもそもクルシュカルのアルゴリズムを実行する必要すらないことがわかる．そのようなグラフに対しては，単に「no；その重みを持つ極大木は存在しない」と答えるだけでよいのである．

また，クルシュカルのアルゴリズムの実行時間の上界を，上では$O(e(m+e))$のようにmとeの関数として求めたが，我々は，遠慮がちに，mとeをそれらより大きめの値であるnで置き換えて，実行時間はnの関数として$O(n(n+n))$，すなわち，$O(n^2)$である，といってもよい．実は，クルシュカルのアルゴリズムをもう少し工夫をこらした方法で実行すると$O(n\log n)$の実行時間しか要しないことも知られている．しかし，ここではそのような改善方法に深入りすることはしない．

もちろん，我々は計算のモデルとしてテューリング機械を使用する．一方，上に説明したアルゴリズムは，配列やポインタなどの便利なデータ構造を備えたプログラミング言語によって実現されることを意図して記述されている．しかし，以下に記すように，上に述べたクルシュカルのアルゴリズムを多テープTM上に移植し$O(n^2)$ステップで実行するようにすることが可能である．補助テープはいくつかの異なる目的のために用いられる．

(1) 1本のテープを，すべての頂点と，各時点における各頂点の連結成分番号との表を記憶するために用いる．この表の長さは$O(n)$である．

(2) 入力テープ上の辺を一つずつ調べて行きながら，まだ「使用済」の印がつけられていない辺の重みの中で最小の重みを求めるが，1本のテープは，

この重みを保持するために用いられる．アルゴリズムの繰り返しの各回ごとに，その時点で残っている辺の中で重みが最小の辺であるとして選択された辺に印をつけておくために，入力テープの第 2 トラックを用いる．最小の重みの，印がついていない辺を探していく作業は $O(n)$ 時間で実行することができる．各辺はただ一度だけ見ればよく，重みの比較は 2 進数を右から左へ 1 方向に見ていくだけでできるからである．

(3)　繰り返しの手順の各回ごとに，一つの辺が選択されたら，その二つの頂点（端点）を 1 本のテープ上に置く．これらの 2 頂点の連結成分を求めるため，頂点と連結成分の表を探索する．この作業は $O(n)$ 時間で実行できる．

(4)　それまで連結されていなかった二つの連結成分を連結する辺が見つかったとき，併合すべき二つの連結成分，i と j，を保持するため，1 本のテープを用いる．次に，頂点と連結成分の表を調べていき，連結成分の番号が i であるすべての頂点について連結成分の番号を j に変える．この作業も $O(n)$ 時間で実行することができる．

以上の考察により，繰り返しのステップの 1 回分を多テープ TM 上で $O(n)$ 時間で実行することができる，という議論が完成する．繰り返しの回数 e は高々 n であるので，多テープ TM 上でも $O(n^2)$ 時間あれば十分であると結論することができる．さてここで，定理 8.10 を思い出していただきたい．そこでは，多テープ TM によって s ステップで実行できることは 1 テープ TM によって $O(s^2)$ ステップで実行できる，ということを述べた．したがって，多テープ TM によって $O(n^2)$ ステップで実行できるならば，同じことを $O((n^2)^2) = O(n^4)$ ステップで実行できるような 1 テープ TM を作ることができる．すなわち，我々の結論は次の通りである．MWST 問題の “yes/no” 版，すなわち，「グラフ G は重みの合計が W 以下の MWST を持つか」という問題はクラス $\mathcal{P}$ に属す．

10.1.3　非決定性多項式時間

実行不能性の学習における一つの基礎的な問題のクラスは，多項式時間で計算を終了する非決定性 TM によって解くことのできる問題のクラスである．形式的に言うと次のようになる．言語 L がクラス $\mathcal{NP}$（非決定性 (**n**ondeterministic) 多項式 (**p**olynomial)）に属すのは，非決定性 TM M と，多項式の時間計算量

非決定性の受理方式の変形

我々は，(多項式時間の) NTM を定義する際，すべての枝分かれの路に沿う計算に対し，それが受理に至るものであろうとなかろうと，必ず，多項式時間で停止することを要求した．これに対し，多項式時間の限界 $T(n)$ を受理に導く路だけに課すような定義の方式もあり得る．すなわち，$\mathcal{NP}$ というクラスを，ある多項式 $T(n)$ に対し，受理する場合には受理に至る路のうち少なくとも一つの路は高々$T(n)$ 回の動作から成る路である，という NTM によって受理される言語の全体として定義する．

しかしながら，もしそのような方式を採用したとしても，得られる言語のクラスは同じである．なぜならば，受理する限りは必ず $T(n)$ 回以下の動作の後に受理するということがわかっているとしたら，M を修正して，テープの別のトラック上でステップ数を $T(n)$ まで数え上げていき，受理せずに $T(n)$ を超えるところまで数えたら，その時点で，受理せずに停止するようにすることができるからである．このように修正された M のステップ数は $O(T^2(n))$ になるかもしれない．しかし，$T(n)$ が多項式であるならば $T^2(n)$ も多項式であることに変わりはない．

実際，我々は，クラス $\mathcal{P}$ の定義も同様に，ある多項式 $T(n)$ に対し，(受理する場合は) 時間 $T(n)$ 以内に受理する，という TM によって受理される言語の全体として与えることもできた．このような TM は，受理しない入力に対しては，停止しないこともあり得る．しかし，NTM に対するものと同様の構成により，ステップ数を $T(n)$ まで数え，その限界値を超えたら停止するように DTM を修正することができる．この修正された DTM の実行時間は $O(T^2(n))$ である．

$T(n)$ で，$L = L(M)$ であり，かつ，長さ n の入力が与えられたとき M の動作の系列で $T(n)$ ステップを超えるものはない，という条件を満たすものが存在するときである．

すぐにわかることは，$\mathcal{P} \subseteq \mathcal{NP}$ が成り立つことである．これは，すべての決定性 TM は複数個 (または 0 個) の動作の選択肢をまったく持つことのない非決定性 TM とみなすことができる，という事実による．これに対し，$\mathcal{NP}$ には $\mathcal{P}$ に属さない問題が多く含まれているように思われる．その直観的な理由は，多項式時間で動く NTM は，与えられた問題に対する解の候補として指数関数の個数のものを推測し，それぞれについて「並列的に」多項式時間で解であるかどうかを確認することができる能力を持つ，というところにある．

- 数学における最も深い未解決問題の一つが，$\mathcal{P} = \mathcal{NP}$ の真偽を定めよ，という問題である．すなわち，NTM によって多項式時間でできることはすべて，多項式の次数は高くなるかもしれないが，DTM によっても多項式時間でできるかどうかを問う問題である．

10.1.4 $\mathcal{NP}$ に属す問題の例：巡回セールスマン問題

$\mathcal{NP}$ の「能力」に対する感覚を養うため，$\mathcal{NP}$ には属すが，$\mathcal{P}$ には属しそうにない問題の例を一つ取り上げて考察してみよう．それは，**巡回セールスマン問題** (Traveling Salesman Problem (TSP)) と呼ばれる問題である．TSP への入力は MWST への入力と同じであり，図 10.1 に示すように，辺が整数の重みを持つグラフと重みの限界値 W との対である．問われるのは，与えられたグラフに重みの和が W 以下の「ハミルトン閉路」が存在するか，という質問である．なお，**ハミルトン閉路** (Hamilton circuit) とは，すべての頂点がちょうど 1 回現れるような一つの閉路を構成する辺の集合である．ハミルトン閉路に含まれる辺の個数はグラフの中の頂点の個数に等しいことに注意されたい．

例 10.3 図 10.1 のグラフは，ハミルトン閉路をただ一つだけ持つ．閉路 $(1, 2, 4, 3, 1)$ がそれである．この閉路の重みの和は $15 + 20 + 18 + 10 = 63$ である．したがって，$W \geq 63$ 以上ならば答は "yes" であり，$W < 63$ ならば答は "no" である．

ただし，頂点数が 4 のグラフに対する TSP は，あまりにも簡単すぎてほとんど参考にならない．それは，閉路の出発点の違いや向きの違いを無視すれば，異なるハミルトン閉路は高々2 個しか存在しないからである．これに対し，頂点数が m のグラフでは異なる閉路の個数は $O(m!)$ にまで増大する．m の階乗 $m!$ は，任意の定数 c に対し，2^{cm} よりも大きくなる． ▨

TSP を解く方法は，どんな方法であろうとも，本質的にすべての閉路を取り上げ，それらの重みの和を計算するという作業を含むように見える．少し賢さを発揮すれば，明らかに悪い選択肢は最初から考慮の範囲から除外しておくことができるかもしれない．しかし，どんなに工夫を凝らしたとしても，与えられた重さの限界値 W 以下の閉路は存在しない場合，または，存在する場合であっても，閉路を調べていく順序が運の悪い順序である場合には，存在しない

という結論を出すまでに，または，解を見つけるまでに，指数関数にのぼる個数の閉路を調べなければならないように思われる．

これに対して，仮に，非決定性のコンピュータを使うことができるとすれば，我々はまず，頂点の順列を推測し，次に，その順に頂点をたどっていくことによって閉路の重みの和を計算することができる．非決定性で動くコンピュータが現実に存在すれば，入力列の長さを n とするとき，どの枝分かれの路も $O(n)$ ステップより多くを必要とすることはない．多テープ NTM では，順列の推測は $O(n^2)$ ステップで行うことができ，重みの和のチェックも同程度の時間で行うことができる．したがって，1 テープ NTM を用いると，TSP は高々$O(n^4)$ 時間で解くことができる．よって，TSP は $\mathcal{NP}$ に属すことが結論される．

10.1.5 多項式時間還元

問題 P_2 は多項式時間では解けない（すなわち，P_2 は $\mathcal{P}$ に属さない）ことを証明するために我々がとる主要な方法は，$\mathcal{P}$ に属さないことがすでに知られているある問題 P_1 に対し，P_1 を P_2 に還元することである[2)]．この考え方はすでに図 8.7 に示したが，図 10.2 に再録しておく．

「P_2 が $\mathcal{P}$ に属すならば，P_1 も $\mathcal{P}$ に属す」という命題を証明したいとする．我々はすでに P_1 は $\mathcal{P}$ に属さないことを主張しているので，この命題が証明さ

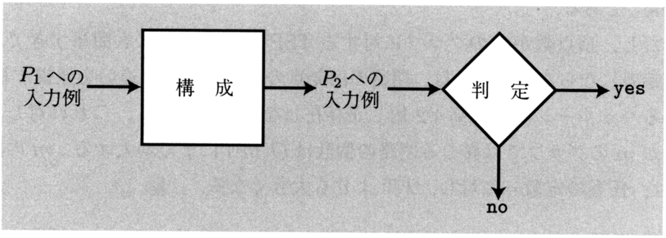


図 10.2 変形の様子を示す図

2) この文章には少々誤りがある．実際には，P_1 は $\mathcal{P}$ に属さないことを**仮定する**だけである．この仮定は，10.1.6 項で導入される *NP* 完全という概念を用いて，P_1 は「*NP* 完全」であるという形で与えられる．これは $\mathcal{P}$ に属さないことを示す非常に強い証拠と考えられている．その仮定のもとで，P_2 もまた「*NP* 完全」であることを証明し，それによって，P_2 は $\mathcal{P}$ に属さないことを同程度に強力に示すのである．

れれば，P_2 が $\mathcal{P}$ に属さないことも同様に主張できることになる．ただし，図10.2 で「構成」とラベルづけされた部分に対し，単に，その部分を処理するアルゴリズムの存在を示すだけでは，目的の命題を証明するのに十分ではない．

たとえば，P_1 の長さ m の入力例が与えられたとき，アルゴリズム「構成」によって長さ 2^m の出力列が作り出され，その文字列が P_2 を解く仮想的な多項式時間アルゴリズム「判定」に入力として与えられたとする．その判定アルゴリズムは，たとえば，$O(n^k)$ 時間で処理を行うものであるとすると，長さ 2^m の入力に対し，判定に要する時間は $O(2^{km})$ である．これは m に関する指数時間である．したがって，P_1 に対するこの判定アルゴリズムは，長さ m の入力に対し，m に関する指数の時間を要することになる．この事実は，P_2 は $\mathcal{P}$ に属し P_1 は $\mathcal{P}$ に属さない，という状況とまったく矛盾するものではない．

また，たとえ，P_1 の入力例から P_2 の入力例を構成するアルゴリズムが常に入力のサイズに関する多項式の長さ程度の入力例を生成するとしても，やはり，期待する結論には到達できずに終わることがあり得る．たとえば，生成される P_2 の入力例は P_1 の入力例と同じ長さの m であるとする．しかし，構成のアルゴリズム自体の計算時間が，たとえば $O(2^m)$ のように，m に関する指数であることがあり得る．このような場合，長さ n の入力に対し多項式時間 $O(n^k)$ を要する P_2 に対する判定アルゴリズムの存在からは，単に，長さ m の入力に対し $O(2^m + m^k)$ 時間を要する P_1 に対する判定アルゴリズムの存在が導かれるだけである．この計算時間の上界値は，P_2 への還元のための計算を，その結果得られる入力例について P_2 を解くための計算と同様に，考慮に入れている．ここでもまた，P_2 は $\mathcal{P}$ に属すが P_1 は $\mathcal{P}$ に属さない，という状況が可能となる．

P_1 から P_2 への還元に関して課すべき適切な条件は，還元の計算に要する時間は入力の長さに関する多項式の時間であるという条件である．長さ m の入力に対する還元に要する時間が $O(m^j)$ である場合には，P_1 の出力列の長さが実行されるステップ数を超えることはあり得ないので，P_1 の出力列の長さは，ある定数 c に対し，高々cm^j であるということに注意されたい．このような条件のもとでならば，P_2 が $\mathcal{P}$ に属すならば，P_1 も $\mathcal{P}$ に属すことを証明することができる．

証明のため，長さ n の文字列に対する P_2 における判定を時間 $O(n^k)$ で行うことができるとする．このとき，長さ m の文字列に対する P_1 における判定は

時間 $O(m^j + (cm^j)^k)$ で行うことができる．項 $O(m^j)$ は還元のために要する時間であり，項 $O((cm^j)^k)$ は還元の結果得られる P_2 への入力について判定をするための時間である．式を書き直すと，P_1 は時間 $O(m^j + cm^{jk})$ で解くことができることがわかる．ここで，c, j, k はすべて定数であるので，この所要時間は m に関する多項式であり，P_1 は $\mathcal{P}$ に属すことが結論される．

このように，実行不能性の理論では，多項式時間還元だけを用いることにする．P_1 から P_2 への還元は，P_1 の入力例の長さに関するある多項式の時間で実行できるとき，**多項式時間還元** (polynomial-time reduction) であるという．この定義の帰結として，P_2 の入力例の長さは P_1 の入力例の長さに関する多項式の長さとなることに注意する．

10.1.6 *NP* 完全問題

次に，$\mathcal{NP}$ には属すが $\mathcal{P}$ には属さない問題の候補として，もっともよく知られている問題の族を見ていくことにしよう．

L を言語（問題）とする．L が次の条件を満たすとき，L は ***NP* 完全** (*NP*-complete) であるという．

(1) L は $\mathcal{NP}$ に属す．

(2) $\mathcal{NP}$ に属す任意の言語 L' に対し，L' から L への多項式時間還元が存在する．

NP 完全問題の一つの例には，あとで見るように，10.1.4 項で紹介した巡回セールスマン問題がある．$\mathcal{P} \neq \mathcal{NP}$ が成り立つように思われることから，すべての *NP* 完全問題は $\mathcal{NP} - \mathcal{P}$ に属すと思われるので，我々は，ある問題に対する *NP* 完全性の証明を，その問題が $\mathcal{P}$ に属さないことの証明のようにみなすことが多い．

NP 完全であることを証明する最初の問題は，SAT（ブール式に対する充足可能性問題）と呼ばれる問題である．SAT の *NP* 完全性の証明は，多項式時間 NTM によって受理される言語はすべて SAT への多項式時間還元を持つことを示すことによってなされる．しかし，一旦ある問題が *NP* 完全であることが証明されたのちは，他の問題が *NP* 完全であることを別の手法によって証明することができる．すなわち，多項式時間還元を用いて，*NP* 完全であることが知られている問題を目的の問題へ還元することによって，目的の問題の *NP* 完全

NP 困難問題

たくさんある問題の中には大変難しい問題もある．ある問題 L に対しては，*NP* 完全性の定義の条件 (2)（$\mathcal{NP}$ に属す任意の言語は多項式時間で L に還元可能である）は証明できるが，条件 (1)（L は $\mathcal{NP}$ に属す）を証明することはできない．L が条件 (2) を満たすとき，L は ***NP* 困難** (*NP*-hard) であるという．我々はこれまで,「実行不能」という用語をやや曖昧な意味で，指数時間を必要とするようにみえる問題を指すために用いてきた.「実行不能」という言葉を「*NP* 困難」を意味するために用いることが一般に受け入れられている．ただし，原理的には，厳密な意味では *NP* 困難でない問題で，指数時間を必要とする問題が存在することはあり得る．

L が *NP* 困難であることの証明は，L が指数時間またはそれ以上の時間を必要とする問題であることを示唆するのに十分である．しかし，L が $\mathcal{NP}$ に属さない場合には，L の難しさが必ずしもすべての *NP* 完全問題の難しさを支持するものとはなり得ない．$\mathcal{P} = \mathcal{NP}$ であるが，その一方，L は指数時間を必要とするということは当然起こり得る．

性を証明するのである．何故そのような還元により対象となる問題が *NP* 完全であることが証明されたことになるのか，その理由を次の定理に示す．

定理 10.4 P_2 は $\mathcal{NP}$ に属すとする．P_1 が *NP* 完全であり，P_1 から P_2 への多項式時間還元が存在するならば，P_2 も *NP* 完全である．

証明 $\mathcal{NP}$ に属す任意の言語 L は P_2 に多項式時間還元可能であることを示せばよい．L から P_1 への多項式時間還元が存在することはわかっている．この還元に要する時間を多項式時間 $p(n)$ であるとする．したがって，L に属す長さ n の文字列 w は，P_1 に属し長さが高々$p(n)$ の文字列 x に変換される．

P_1 から P_2 への多項式時間還元が存在することも仮定されている．この還元に要する時間はある多項式時間 $q(m)$ であるとしよう．そうすると，この還元により x は，高々$q(p(n))$ の時間で，P_2 に属すある文字列 y に還元される．したがって，w から y への変形は高々$p(n) + q(p(n))$ の時間で行うことができる．これは n に関する多項式である．すなわち，L は P_2 に多項式時間還元可能であると結論することができる．L は $\mathcal{NP}$ に属す言語であればどんな言語でもよかったので，$\mathcal{NP}$ に属すすべての言語から P_2 に多項式時間還元を行うことができることが示された．以上より，P_2 は *NP* 完全である． ▨

$\boldsymbol{NP}$ 完全性の別の定義

NP 完全性を研究する目的の一つは，実は，定理 10.5 にある．すなわち，クラス $\mathcal{P}$ に属すことが直ちに $\mathcal{P}=\mathcal{NP}$ を導くような問題 P の特定である．我々が採用した「NP 完全」の定義は，カープ (R. Karp) がこの主題に関する基本的な論文の中で最初に用いたものなので，**カープ完全性** (Karp-completeness) と呼ばれることがある．この定義は，我々が定理 10.5 を満たすと信ずるに足る理由を持つ，あらゆる問題を捉えるのに適している．しかし，これとは別のもっと広い NP 完全性の概念で，やはり定理 10.5 の主張を可能にするようなものも存在する．

たとえば，クックは，この主題の出発点となった最初の論文の中で，問題 P が「NP 完全」であることを次のように定義した．問題 P に対する**神託** (oracle)，すなわち，任意に与えられた文字列に対し，それが P に属すかどうかという質問に対する答を 1 単位時間で与える機構，の利用が許されれば，$\mathcal{NP}$ に属す任意の言語を多項式時間で認識することができるとき，P は「NP 完全」であるという．この型の NP 完全性は，**クック完全性** (Cook-completeness) と呼ばれる．カープ完全性は，ある意味で，神託への質問が 1 回だけに限定された，クック完全性の特殊な場合であるとみなすことができる．ただし，クック完全性では，解の反転（yes と no の逆転）も可能となる．たとえば，神託に対して質問をし，神託から答を得たらその逆を答えることが可能である．クックの定義から導かれる一つの帰結は，NP 完全問題の補集合はまた NP 完全である，というものである．一方，我々が採用しているような，もう少し制限された形のカープ完全性を用いると，(カープの意味の）NP 完全問題とその補集合との間の大きな違いを明確にすることが可能になる．これについては，11.1 節で詳述する．

NP 完全問題に関して証明すべき事柄で，重要な定理がもう一つある．それは，NP 完全問題のうちのどれか一つでも $\mathcal{P}$ に属すとすれば，$\mathcal{NP}$ に属すすべての問題が $\mathcal{P}$ に属す，というものである．我々は，$\mathcal{NP}$ に属す問題で $\mathcal{P}$ には属さないものが数多く存在することを強く信じる．したがって，その信念の一つの帰結として，ある問題が NP 完全であることを示す証明は，その問題は多項式時間のアルゴリズムを持たず，したがって，コンピュータで解くことのできる良い解を持たないということの証明と同等なものとみなすことになる．

定理 10.5　ある NP 完全問題 P が $\mathcal{P}$ に属すならば，$\mathcal{P}=\mathcal{NP}$ が成り立つ．

証明　P は NP 完全であり，かつ，$\mathcal{P}$ に属すとする．そうすると，$\mathcal{NP}$ に属すすべての言語 L は，多項式時間で P に還元される．このとき，10.1.5 項で論じたように，P が $\mathcal{P}$ に属すならば，L も $\mathcal{P}$ に属す．▨

10.1.7　10.1 節の演習問題

♣　**問 10.1.1**　図 10.1 のグラフで辺の重みを次のように変更したとする．新しいグラフに対する MWST を求めよ．

*a)　辺 $(1,3)$ の重み 10 を 25 に変える．

b)　辺 $(2,4)$ の重みを 16 に変える．(辺 $(1,3)$ の重みはもとのままとする.)

♣　**問 10.1.2**　図 10.1 のグラフを，頂点 1 と頂点 4 の間に重み 19 の辺を付加するように変更したとする．このグラフに対し，重みが最小のハミルトン閉路を求めよ．

♣*! **問 10.1.3**　計算時間 $O(n^{\log_2 n})$ の決定性の解を持つ NP 完全問題が存在したとする．この関数 $n^{\log_2 n}$ は，多項式と指数関数の中間に位置し，そのどちらの範疇にも属さないことに注意せよ．このとき，$\mathcal{NP}$ に属す任意の問題の時間計算量に関してどんなことが言えるかを考えよ．

♣*! **問 10.1.4**　サイズが m の n 次元超立方体を考え，その格子点の全体を頂点集合とするグラフを考える．すなわち，頂点の集合は，$\{1,2,\ldots,m\}$ に属す任意の i_j に対する，ベクトル $(i_1,i_2,\ldots,i_n)$ の全体である．二つの頂点の間に辺があるのは，それらがただ一つを除くすべての位置で成分が相等しく，異なる一つの位置の成分は差が 1 であるときであり，またそのときに限る．たとえば，$n=2$, $m=2$ の場合は正方形であり，$n=3$, $m=2$ の場合は立方体である．さらに，$n=2$, $m=3$ の場合は図 10.3 に示すグラフとなる．これらのグラフのうちのあるものはハミルトン閉路を持ち，また，あるものはハミルトン閉路を持たない．たとえば，正方形は明らかにハミルトン閉路を持つ．立方体もハミルトン閉路を持つが，正方形ほど自明ではない．たとえば，$(0,0,0)$, $(0,0,1)$, $(0,1,1)$, $(0,1,0)$, $(1,1,0)$, $(1,1,1)$, $(1,0,1)$, $(1,0,0)$, $(0,0,0)$, がハミルトン閉路の一つの例である．図 10.3 のグラフはハミルトン閉路を持たない．

a)　図 10.3 のグラフはハミルトン閉路を持たないことを証明せよ．**ヒント：**ハミルトン閉路が存在するとして，その閉路が中央の頂点を通過するときにどんなことが起こるかを考えてみよ．グラフの一部をハミルトン閉路から切り離さずにすませるには，その閉路は（中央の頂点に）どこから入ってきて，どこへ出ていけばよいだろうか．

b)　ハミルトン閉路が存在するような n と m の値の組をすべて求めよ．

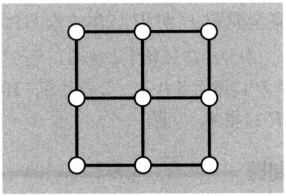


図 10.3　$n = 2,\ m = 3$ の場合のグラフ

♣ **!問 10.1.5**　有限個の文字から成るあるアルファベットを用いて，文脈自由文法を符号化するとする．次の二つの言語を考える．

(1)　$L_1 = \{(G, A, B) \mid G$ は（符号化された）CFG である．A と B は G の（符号化された）変数であり，A から導出される終端列の集合と B から導出される終端列の集合は相等しい $\}$

(2)　$L_2 = \{(G_1, G_2) \mid G_1$ と G_2 は（符号化された）CFG であり，$L(G_1) = L(G_2)$ である $\}$

次の問に答えよ．

*a)　L_1 は L_2 に多項式時間還元可能であることを示せ．

b)　L_2 は L_1 に多項式時間還元可能であることを示せ．

*c)　L_1 と L_2 が NP 完全であるかどうかということについて，(a) と (b) とから言えることを記せ．

♣ **!問 10.1.6**　言語の族の場合と同様に，$\mathcal{P}$ と $\mathcal{NP}$ はそれぞれある種の閉包性を持つ．$\mathcal{P}$ は，以下の各々の演算のもとで閉じていることを示せ．

a)　反転

* b)　（集合の）和

*!c)　連接

!d)　（スター）閉包

e)　逆準同型写像

* f)　補集合

♣ **!問 10.1.7**　$\mathcal{NP}$ もまた，$\mathcal{P}$ のためにあげた問 10.1.6 の各々の演算のもとで閉じている．ただし，(f) の補集合の演算だけは（恐らく）例外である．現時点では，$\mathcal{NP}$ が補集合の演算のもとで閉じているかどうかはまだわかっていない．これについては，11.1 節で詳しく述べる．$\mathcal{NP}$ は，問 10.1.6 の (a) から (e) の各々の演算のもとで閉じていることを証明せよ．

10.2　最初の NP 完全問題

さて，いよいよ，最初の NP 完全問題の紹介に移ろう．最初に取り上げる問題 —— ブール式が充足可能であるかどうかを問う問題 —— が NP 完全であることは，任意の非決定性多項式時間 TM によって受理される言語から充足可能性問題への還元を具体的に与えることによって証明される．

10.2.1　充足可能性問題

ブール式 (boolean expression) は次のものから構成される．

(1)　変数．　変数の値はブール値，すなわち，1（真，true）または 0（偽，false）である．

(2)　2 項演算 $\wedge$ と $\vee$．　二つの式（オペランド）に対する論理積 AND と論理和 OR を表す．

(3)　1 項演算 $\neg$．　論理否定を表す．

(4)　括弧．　演算子の優先順位を変更する必要があるときに，演算子とオペランドをグループ分けするために用いる．とくに指定しないときの演算子の優先順位は，高い順に次の通りとする．$\neg$, $\wedge$, $\vee$.

例 10.6　$x \wedge \neg(y \vee z)$ はブール式の一つの例である．部分式 $y \vee z$ は，変数 y または変数 z の少なくとも一方の値が真であるとき真になり，y と z がともに偽であるとき偽になる．それより大きな部分式 $\neg(y \vee z)$ は，$y \vee z$ が偽であるとき，すなわち，y と z がともに偽であるとき，またそのときに限り，真になる．y と z のいずれか一方，または，両方が真のとき，$\neg(y \vee z)$ は偽になる．

最後に，上の式の全体を考えてみる．上の式は，二つの部分式に論理積 AND を施したものなので，両方の部分式が真であるとき，またそのときに限り，真になる．言いかえると，$x \wedge \neg(y \vee z)$ が真になるのは，x が真であり，y が偽であり，かつ，z が偽であるときであり，またそのときに限る．　▨

ブール式 E に対する**真理値割り当て** (truth assignment) とは，E に含まれる各変数に対し真または偽の値を割り当てることである．真理値割り当て T のもとでの式 E の**値** (value) とは，E の中に現れる各変数 x を T が x に割り当てる値 $T(x)$（真または偽）で置き換えた式を評価して得られる結果である．真

理値割り当て T のもとでの式 E の値を $E(T)$ と表す．

真理値割り当て T がブール式 E を**充足する** (satisfy) とは，$E(T) = 1$ となること，すなわち，真理値割り当て T が式 E を真にすることである．ブール式 E が**充足可能** (satisfiable) であるとは，E を充足する真理値割り当て T が少なくとも一つ存在することである．

例 10.7 例 10.6 に示した式 $x \wedge \neg(y \vee z)$ は充足可能である．$T(x) = 1$, $T(y) = 0$, $T(z) = 0$ によって定義される真理値割り当て T はこの式を充足することをすでに見ている．この真理値割り当ては式の値を真 (1) にする．我々は，T はこの式を充足する**ただ一つ**の真理値割り当てであることも知っている．三つの変数に対する値の組合せのうち，その他の 7 組は式の値を偽 (0) にするからである．

もう一つの例として，$E = x \wedge (\neg x \vee y) \wedge \neg y$ という式を考えてみよう．E は充足可能ではないことを示したい．この式には変数は二つしか含まれていないので，真理値割り当ての個数は $2^2 = 4$ だけである．したがって，四つの真理値割り当てを順に試し，すべての割り当てに対して E が値 0 をとることを検証すればよいが，それは容易なことである．しかし，それとは別に，次のように論証を進めることもできる．E が真になるのは $\wedge$ によって結ばれた三つの項の値がすべて真のときである．そのためには,（第 1 項である）x が真でなければならず，また,（第 3 項である）$\neg y$ も真でなければならない．しかし，そのような真理値割り当てのもとでは，第 2 項 $\neg x \vee y$ は偽になる．したがって，E を真にすることはできず，E は充足可能ではない．

充足可能な真理値割り当てをただ一つ持つ式の例と，充足可能な真理値割り当てを一つも持たない式の例とを見てきた．これに対し，充足可能な真理値割り当てを二つ以上持つ式の例もたくさんある．簡単な例として，$F = x \vee \neg y$ という式を考えてみよう．F の値は，次の三つの真理値割り当てに対して 1 となる．

(1) $T_1(x) = 1$; $T_1(y) = 1$.

(2) $T_2(x) = 1$; $T_2(y) = 0$.

(3) $T_3(x) = 0$; $T_3(y) = 0$.

F の値が 0 になるのは，第 4 の真理値割り当て $x = 0, y = 1$ に対してのみである．F を真にする真理値割り当てが存在するので，F は充足可能である． ▨

次の問題を**充足可能性問題** (satisfiability problem) という．

- ブール式が与えられたとき，それが充足可能であるかどうかを判定せよ．

充足可能性問題を **SAT** と称する．言語として記述すると，問題 SAT とは，充足可能な（符号化された）ブール式の全体から成る集合である．ブール式に対する正しい符号化列となっていない文字列や，充足可能でないブール式に対する符号化列となっている文字列は，言語 SAT に属さない．

10.2.2 SAT に対する入力例の表現

ブール式の中で用いられる記号（文字）は，$\wedge$，$\vee$，$\neg$，左括弧，右括弧，そして，変数を表す記号である．ブール式の充足可能性は，変数の名前のつけ方には依存しない．依存するのは，相異なる 2 個所に出現している変数に対し，それらが同じ変数であるのか，それとも，相異なる変数であるのか，という点だけである．したがって，我々は，変数は常に $x_1, x_2, \ldots$ であると仮定してもさしつかえない．ただし，具体例を示すような場合には，x_i の他に，相変わらず y や z のような変数名も使うことにする．我々はまた，(必要ならば変数の名前をつけかえて,）変数の添字の番号を，飛び飛びに使用することなく，小さい方から順に使用することを仮定する．たとえば，我々は，同じ式の中で x_1 から x_4 までをすべて使っている場合を除いて，x_5 を使うことはしない．

ブール式の中に現れる記号は，可能性としては，無限個あり得る．そこで，任意に大きな個数の変数を含む式を，固定された有限集合であるアルファベットを用いて符号化しなければならず，そのための工夫が必要になる．しかし，これはすでにおなじみの事柄である．この懸案を解決してはじめて，我々は，SAT を一つの「問題」，すなわち，充足可能なブール式に対する符号化列から成る，固定されたアルファベット上の言語として扱うことができるようになる．我々が使用する記号は，次の通りである．

(1) 記号 $\wedge$，$\vee$，$\neg$，(，) は，そのまま，それ自身によって表現される．

(2) 変数 x_i は，記号 x と，それに続く i の 2 進数表示である 0 と 1 の列とによって表現される．

したがって，SAT 問題，あるいは，SAT 言語に対するアルファベットは 8 個の記号だけから成る．SAT のすべての入力例は，この固定された有限の大きさのアルファベットにより構成される文字列である．

例 10.8 例 10.6 に示した式 $x \wedge \neg(y \vee z)$ を考えてみよう．この式を符号化するための最初の手順は，各変数を添字つきの x で置き換えることである．変数は 3 個あるので，x_1, x_2, x_3 を使う必要がある．x, y, z のどれを x_1, x_2, x_3 のどれで置き換えるかは自由である．ここでは，$x = x_1$, $y = x_2$, $z = x_3$ としよう．そうすると，与えられた式は，$x_1 \wedge \neg(x_2 \vee x_3)$ となる．この式に対する符号化列は，次の通りである．

$$x1 \wedge \neg(x10 \vee x11)$$

▨

符号化されたブール式の長さは，変数の 1 回の出現を 1 と数えたときの，式の中の変数位置の個数と，概ね，同じであることに注意されたい．差が生じる理由は，仮に，式が m 個の変数位置を持つとすると，式は $O(m)$ 個の変数を持つことが可能であり，したがって，一つの変数の符号化列が $O(\log m)$ 個の記号から成ることがあり得るからである．したがって，m 個の変数位置の長さを持つ式に対し，その符号化列は $n = O(m \log m)$ 個の記号から成る文字列になることがあり得る．

しかし，m と $m \log m$ の差は，明らかに多項式で抑えられる程度の差である．したがって，我々の関心の対象が，与えられた問題が入力列の長さに関する多項式の時間で解けるか否か，というものである限り，式を符号化して得られる列の長さと元の式に含まれる変数位置の個数とを敢えて区別する必要はない．

10.2.3 SAT 問題の *NP* 完全性

SAT は *NP* 完全であることを主張する「クックの定理」を証明しよう．一般に，ある問題が *NP* 完全であることを証明するには，まず，その問題が $\mathcal{NP}$ に属すことを示すことが必要であり，次に，$\mathcal{NP}$ に属すすべての言語をその問題に還元することができることを示すことが必要である．多くの場合，この第 2 の部分は，ある別の *NP* 完全問題から対象の問題への多項式時間還元を与え，

さらに定理 10.4 に訴えることにより，証明される．しかし，現時点では，別の NP 完全問題から SAT への還元を示そうにも，我々はまだ一つも NP 完全問題を知らない．したがって，現時点で採用可能なただ一つの方針は，$\mathcal{NP}$ に属す任意の問題を SAT に還元することである．

定理 10.9 **（クックの定理）** SAT は NP 完全である．

証明 証明の第 1 の部分は，SAT が $\mathcal{NP}$ に属すことを示すことである．この部分は簡単である．

(1) NTM の非決定性の機能を用いて，与えられたブール式 E に対する真理値割り当て T を推測する．符号化された列 E の長さが n であるとき，この処理に要する時間は，多テープ NTM 上で $O(n)$ 時間で十分である．この NTM は動作の選択を何度も行う．推測の処理が完了するときに到達可能な相異なる ID の個数は 2^n 程度にまで達することに注意されたい．枝分かれの各々の路が，相異なる真理値割り当ての推測に対応する．

(2) 真理値割り当て T に対し，ブール式 E を評価する．その結果，$E(T) = 1$ であれば，入力を受理する．この部分の計算は，決定性の計算であることに注意されたい．NTM の別の推測による路では受理に導かれないかもしれないが，そのことは結果に何の影響も与えない．定義より，式を充足する真理値割り当てが一つでも見つかれば，NTM は入力を受理するからである．

式の評価の部分は，多テープ NTM 上で $O(n^2)$ 時間で容易に実行することができる．したがって，多テープ NTM による SAT の認識に要する時間は，全体で $O(n^2)$ 時間である．これを 1 テープ NTM に変換すると，必要な計算時間は高々2 乗されるだけなので，$O(n^4)$ 時間が 1 テープ NTM 上での実行時間の上界となる．

さて次に，第 2 の部分の証明に移ろう．この証明は第 1 の部分の証明よりはるかに難しい．証明すべきことは，L を $\mathcal{NP}$ に属す任意の言語とするとき，L から SAT への多項式時間還元が存在するということである．ある 1 テープ NTM M と多項式 $p(n)$ で，M は長さ n の入力に対し，枝分かれのどの路を通る場合でも，$p(n)$ 以下のステップ数で停止する，という条件を満たすものが存在すると仮定してよい．さらに，定理 8.12 に与えた次のような制限も課すことにする．定理 8.12 では，この制限を DTM に対してのみ証明したが，NTM に対しても同様の方法で証明することができる．すなわち，課すべき制限とは，M は，テープに空白文字を書き込まないことと，テープヘッドを最初のヘッドの位置より左方には動かさないことである．

以上より，M が入力 w を受理し，$|w| = n$ が成り立つとき，M の動作の列で次を満たすものが存在する．

(1) α_0 は入力 w に対する M の初期 ID である．

(2) $\alpha_0 \vdash \alpha_1 \vdash \cdots \vdash \alpha_k$ ただし，$k \le p(n)$ である．

(3) α_k は受理状態を持つ ID である．

(4) 各 α_i は空白以外の文字のみで構成される．(ただし，α_i の末尾の空白文字のみの列の部分を除く．) また，α_i は最初のヘッドの位置 —— 入力列の 1 番左の文字の位置 —— から右方に伸びる列である．

証明の方針をまとめると，次のようになる．

a) 各 α_i は，記号の列 $X_{i0}X_{i1}\cdots X_{i,p(n)}$ として表すことができる．これらの記号のうち一つは状態であり，他はすべてテープ記号である．いつものように，状態の集合とテープ記号の集合は互いに素である（共通部分は空である）と仮定する．したがって，どの X_{ij} が状態であるかを判別することができるし，さらに，テープヘッドがどの位置にあるかも知ることができる．なお，テープ上の左端から $p(n)$ 個の記号よりさらに右方に位置する記号を表示する理由はまったくないこと［したがって，状態を含めた ID の長さは $p(n)+1$ としてよいこと］に注意されたい．M が $p(n)$ 回以下の動作ののちに停止することが保証されている場合には，それらの記号が M の動作に影響を与えることはあり得ないからである．

b) ID の列をブール変数を用いて記述するため，$X_{ij} = A$ という命題を表現するための変数 y_{ijA} を用意する．ここで，i と j はそれぞれ $\{0, 1, \ldots, p(n)\}$ に属す整数であり，A はテープ記号，または，状態である．

c) 「与えられた ID の列は入力 w の受理を表す」という条件を，M が w を高々 $p(n)$ 回の動作の列によって受理するとき，またそのときに限り，充足可能であるようなブール式を書くことによって表現する．充足可能な真理値割り当てとは，ID の列について「真実を告げる」割り当てである．すなわち，y_{ijA} が真になるのは $X_{ij} = A$ であるとき，またそのときに限る．$L(M)$ から SAT への多項式時間還元が正しいものであることを保証するため，計算が次の i), ii), iii) を満たすということと同等のことを主張するようなブール式を書く．

  i) **計算は正しく始まる．** 初期 ID は q_0w とそれに続く空白文字の列とから成る．

  ii) **次に進む動作は正しい．**（動作は TM の規則に正しくしたがう．）各々の ID は，一つ前の ID から M の定められた動作のうちの一つによって導かれる．

  iii) **計算は正しく終わる．** 受理状態を持つ ID が存在する．

ブール式の細部にわたる構成を示す前に，いくつかの事柄について説明をしておく必

要がある．

- 第 1 に，我々はこれまで，ID の記述は，空白文字だけの無限の列が始まる直前までの部分で指定することにしていた．しかし，多項式時間の計算を模倣する際には，すべての ID が同じ長さ，$p(n)+1$，を持つように想定する方が便利である．したがって，一つの ID の後方部分には，一般に，空白文字だけから成る列が存在する．
- 第 2 に，すべての計算が $p(n)$ 回の動作を行う以前に受理状態に入るような場合でも，ちょうど $p(n)$ 回の動作を行う［したがって，$p(n)+1$ 個の ID を持つ］ように仮定しておくと都合がよい．そこで，受理状態を持つ ID はすべて自分自身を（ただ一つの）後継 ID として持つ，と定める．すなわち，受理状態を持つ ID α に対し，$\alpha \vdash \alpha$ という「動作」を認める．こう定めておけば，受理される計算の場合 $\alpha_{p(n)}$ は必ず受理 ID であると仮定することができる．したがって，条件「計算は正しく終わる」を調べるには $\alpha_{p(n)}$ を調べればそれで十分である．

図 10.4 に M の多項式時間の計算の様子を示す．各行は ID の列に対応し，各列は計算の中で用いられる可能性のあるテープのます目を表す．図 10.4 の中のます目の個数は $(p(n)+1)^2$ であることに注意されたい．また，各々のます目に表示される変数の個数は，M だけに依存して定まる有限の値，すなわち，M の状態の個数とテープ記号の個数の和である．

さて，M と w とからブール式 $E_{M,w}$ を構成するためのアルゴリズムを与えることにしよう．目的の式 $E_{M,w}$ の全体の形は $S \wedge N \wedge F$ である．ただし，S, N, F はそれぞれ M について「計算は正しく始まる」,「次に進む動作は正しい」,「計算は正しく終わる」を表すブール式である．

ID	0	1	$\cdots$				$\cdots$	$p(n)$
α_0	X_{00}	X_{01}						$X_{0,p(n)}$
α_1	X_{10}	X_{11}						$X_{1,p(n)}$
α_i				$X_{i,j-1}$	$X_{i,j}$	$X_{i,j+1}$		
α_{i+1}				$X_{i+1,j-1}$	$X_{i+1,j}$	$X_{i+1,j+1}$		
$\alpha_{p(n)}$	$X_{p(n),0}$	$X_{p(n),1}$						$X_{p(n),p(n)}$

図 10.4　セル/ID の並びの構成

計算は正しく始まる　X_{00} は M の初期状態 q_0 でなければならず，X_{01} から X_{0n} までの部分は w でなければならず（ただし，n は w の長さである），さらに，残りの X_{0j} はすべて空白文字 B でなければならない．すなわち，$w = a_1a_2\cdots a_n$ とするとき，

$$\begin{aligned} S \;=\; & y_{00q_0} \wedge y_{01a_1} \wedge y_{02a_2} \wedge \cdots \wedge y_{0na_n} \\ & \wedge y_{0,n+1,B} \wedge y_{0,n+2,B} \wedge \cdots \wedge y_{0,p(n),B} \end{aligned}$$

M を符号化した列と w とが与えられたとき，S を，多テープ TM の第 2 テープ上に $O(p(n))$ 時間で書くことができる．

計算は正しく終わる　受理 ID は際限なく繰り返すと仮定しているので，M によって受理されることと $\alpha_{p(n)}$ が受理状態を含むこととは同値である．M は，入力を受理する場合には必ず $p(n)$ ステップ以内に受理する NTM であると仮定したことを思い出していただきたい．したがって，$j = 0, 1, \ldots, p(n)$ に対し，$X_{p(n),j}$ が受理状態であることを主張する式を F_j とするとき，F_j の全体を OR でつないだ式を F とすればよいことになる．すなわち，$a_1, a_2 \ldots, a_k$ を M の受理状態の全体とするとき，

$$F_j = y_{p(n),j,a_1} \vee y_{p(n),j,a_2} \vee \cdots \vee y_{p(n),j,a_k}$$

とし，さらに，

$$F = F_0 \vee F_1 \vee \cdots \vee F_{p(n)}$$

とすればよい．

ここで，各 F_i の記述に用いられる記号の個数は，M には依存するが，入力 w の長さ n には依存しない定数であることに注意されたい．したがって，F の長さは $O(n)$ である．さらに重要なことは，M の符号化列と w とが与えられたとき，F を書き下すのに要する時間は n に関する多項式であることである．実際，F は多テープ TM により $O(p(n))$ 時間で書くことができる．

次に進む動作は正しい　M の動作が正しいことを確かめる部分は，上の二つに比べ，はるかに難しい部分である．すべての $i = 0, 1, \ldots, p(n) - 1$ に対し，式 N_i を，ID α_{i+1} は ID α_i の次に来ることが可能な ID のうちの一つであるということを保証するように作られた式であるとするとき，このような N_i の全体を AND でつないだ式を N とすればよい．N_i の作り方の説明を始めるにあたり，まず，図 10.4 の中の記号 $X_{i+1,j}$ を見ていただきたい．我々は，$X_{i+1,j}$ を常に以下の事柄から決定することができる．

(1)　$X_{i+1,j}$ のすぐ上の三つの記号：$X_{i,j-1}$, X_{ij}, $X_{i,j+1}$

(2)　これらの三つの記号のうちの一つが α_i の「状態」である場合，NTM M による可能な動作の中からの特定の一つの動作の選択

式 N_i を，$j = 0, 1, \ldots, p(n)$ に対して，式 $A_{ij} \vee B_{ij}$ の全体を $\wedge$ で結びつけたものとする．

- 式 A_{ij} は次のことを主張する．
  - a) α_i の状態は位置 j にある．(すなわち，X_{ij} は状態である．)
  - b) X_{ij} が M の状態であり，$X_{i,j+1}$ がテープヘッドが見ている記号であるとき，記号の列 $X_{i,j-1}X_{ij}X_{i,j+1}$ を列 $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1}$ に変換するような M の動作の選択がある．なお，X_{ij} が受理状態であるときは，全く変化しないという動作の「選択」があること，したがって，そのあとに続く ID はすべて最初に受理状態に入った ID と同じ ID であることに注意されたい．
- 式 B_{ij} は次のことを主張する．
  - a) α_i の状態は X_{ij} から十分遠い位置にあり，$X_{i+1,j}$ に影響を与えることはない．(すなわち，$X_{i,j-1}, X_{ij}, X_{i,j+1}$ はいずれも状態ではない．)
  - b) $X_{i+1,j} = X_{ij}$

このうち，B_{ij} の方が書きやすい．$q_1, q_2, \ldots, q_m$ を M の状態の全体とし，$Z_1, Z_2, \ldots, Z_r$ を M のテープ記号の全体とする．このとき，B_{ij} は次のように表される．

$$
\begin{aligned}
B_{ij} = &(y_{i,j-1,Z_1} \vee y_{i,j-1,Z_2} \vee \cdots \vee y_{i,j-1,Z_r}) \wedge \\
&(y_{i,j,Z_1} \vee y_{i,j,Z_2} \vee \cdots \vee y_{i,j,Z_r}) \wedge \\
&(y_{i,j+1,Z_1} \vee y_{i,j+1,Z_2} \vee \cdots \vee y_{i,j+1,Z_r}) \wedge \\
&((y_{i,j,Z_1} \wedge y_{i+1,j,Z_1}) \vee (y_{i,j,Z_2} \wedge y_{i+1,j,Z_2}) \vee \cdots \vee (y_{i,j,Z_r} \wedge y_{i+1,j,Z_r}))
\end{aligned}
$$

B_{ij} の第 1 行は $X_{i,j-1}$ がテープ記号の一つであることをいう．第 2 行は $X_{i,j}$ がテープ記号の一つであることをいう，第 3 行は同じことを $X_{i,j+1}$ に対していう．最後の行は，$X_{ij} = X_{i+1,j}$ が成り立つことを，すべてのテープ記号 Z を順に取り上げ，X_{ij} と $X_{i+1,j}$ がともに Z_1 である，または，ともに Z_2 である，または，ともに Z_3 である，等々のいずれかが成り立つことを示すことにより主張する．

特別の配慮を必要とする場合が二つある．$j = 0$ の場合と $j = p(n)$ の場合である．前者の場合は変数 $y_{i,j-1,Z}$ が欠けていて，後者の場合は変数 $y_{i,j+1,Z}$ が欠けている．しかし，我々は，テープヘッドについてそれが最初に置かれた位置より左方に行くことはないことを知っているし，また，最初の位置の右方において $p(n)$ 個を超えるます目にまで到達する時間はないことも知っている．したがって，B_{i0} と $B_{i,p(n)}$ からはいくつかの項を削除しておけばよい．その詳細は読者に委ねる．

さて次に，式 A_{ij} について考えよう．これらの式は，図 10.4 の並びの中で 2×3 の長方形に含まれる記号 $X_{i,j-1}$，$X_{ij}, X_{i,j+1}$，$X_{i+1,j-1}$，$X_{i+1,j}$，$X_{i+1,j+1}$ の間のすべての可能な関係を記述するものである．これらの 6 個の変数の各々に対する記号の割り当ては，次の条件を満たすとき，**正当** (valid) であるという．

(1) X_{ij} は状態であり，$X_{i,j-1}$ と $X_{i,j+1}$ はテープ記号である．

(2) $X_{i+1,j-1}$，$X_{i+1,j}$，$X_{i+1,j+1}$ のうち，ちょうど一つが状態である．

(3) $X_{i,j-1}X_{ij}X_{i,j+1}$ から

$$X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1}$$

への変化を可能にする M の動作が存在する．

したがって，6 個の変数への正当な記号の割り当ては複数個あり得る．正当な割り当てを形成する 6 変数の各組に対して一つの項を作り，それらの項を OR で結んだものを A_{ij} とすればよい．

たとえば，M のある一つの動作は $\delta(q, A)$ が (p, C, L) を含むことから生じる動作であるとする．D を M の一つのテープ記号とする．このとき，$X_{i,j-1}X_{ij}X_{i,j+1} = DqA$ かつ $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = pDC$ は，一つの正当な割り当てである．M のこの動作の実行によって生じる ID の中の変化を，この割り当てがどのように反映しているか，という点に注意を払っていただきたい．この変化の可能性を表現する項は，次のようなものである．

$$y_{i,j-1,D} \wedge y_{i,j,q} \wedge y_{i,j+1,A} \wedge y_{i+1,j-1,p} \wedge y_{i+1,j,D} \wedge y_{i+1,j+1,C}$$

次に，$\delta(q, A)$ は (p, C, R) を含むとする．(すなわち，動作は上とほとんど同じであるが，ヘッドが右に動く点が異なる.) これに対応する正当な割り当ては，$X_{i,j-1}X_{ij}X_{i,j+1} = DqA$ かつ $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = DCp$ である．この正当な割り当てを表す項は，次のようになる．

$$y_{i,j-1,D} \wedge y_{i,j,q} \wedge y_{i,j+1,A} \wedge y_{i+1,j-1,D} \wedge y_{i+1,j,C} \wedge y_{i+1,j+1,p}$$

A_{ij} は，すべての正当な項を OR で結んだものである．なお，$j = 0$ と $j = p(n)$ の特別の場合には，$j < 0$ または $j > p(n)$ に対し変数 y_{ijZ} は存在しないという事情を反映するように一部修正を施す必要がある．これは，B_{ij} のときと同様である．最後に，

$$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \cdots \wedge (A_{i,p(n)} \vee B_{i,p(n)})$$

とし，さらに，次のように定める．

$$N = N_0 \wedge N_1 \wedge \cdots \wedge N_{p(n)-1}$$

M が状態やテープ記号をたくさん持つ場合，A_{ij} と B_{ij} は非常に長い式になることがあり得る．しかし，入力 w の長さに関する限り，それらの長さは，あくまでも，定数である．すなわち，それらの大きさは w の長さ n からは独立である．したがって，N_i の長さは $O(p(n))$ であり，N の長さは $O(p^2(n))$ である．さらに重要なことは，N を，多テープ TM の一つのテープ上に，その長さに比例する量の時間で書くことができるということであり，そして，その時間量は w の長さ n に関する多項式の時間であるということである．

クックの定理の証明の結論　これまで，ブール式

$$E_{M,w} = S \wedge N \wedge F$$

の構成を M と w の両方の関数として記述してきたが，実際は，w の内容に依存するのは「計算は正しく始まる」の部分 S だけであり，それも，単純な形（w は初期 ID のテープ上にある）で依存するだけである．残りの二つの部分 N と F は，M と，w の長さ n とだけに依存する．

時間計算量がある多項式時間 $p(n)$ である任意の NTMM に対し，長さ n の入力 w を受け取り $E_{M,w}$ を生成するアルゴリズムを構築することができる．このアルゴリズムの多テープ，決定性 TM 上での実行時間は $O(p^2(n))$ である．さらに，この多テープ TM を，計算時間が $O(p^4(n))$ の 1 テープ TM に変換することができる．このアルゴリズムの出力結果 $E_{M,w}$ は，充足可能であるための必要十分条件は M が w を $p(n)$ 回以下の動作で受理することである，という性質を満たすブール式である．　▨

クックの定理（定理 10.9）の重要性を確認するため，定理 10.5 の定理 10.9 への応用のしかたを見てみよう．SAT が，入力を多項式時間 $q(n)$ で認識する決定性 TM を持つと仮定する．そうすると，多項式時間 $p(n)$ 以内で実行を終

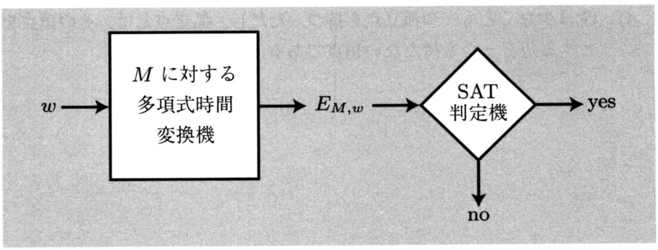


図 10.5　SAT が $\mathcal{P}$ に属すとすると，$\mathcal{NP}$ に属すすべての言語は，この図のように設計される DTM により，$\mathcal{P}$ に属すことになる．

了する NTMM によって受理されるあらゆる言語は，図 10.5 に計算手順の概要が与えられている DTM によって決定性多項式時間で受理されることになる．M に対する入力 w はブール式 $E_{M,w}$ に変換され，そのブール式が SAT 判定機に投入される．この判定機が $E_{M,w}$ に対して出力する答が，そのまま，我々のアルゴリズムに対する答となる．

10.2.4　10.2 節の演習問題

♣ **問 10.2.1**　次の各々のブール式は，充足する真理値割り当てをいくつ持つかを答えよ．また，SAT に属すかどうかを答えよ．

*a)　$x \wedge (y \vee \neg x) \wedge (z \vee \neg y)$

b)　$(x \vee y) \wedge (\neg(x \vee z) \vee (\neg z \wedge \neg y))$

♣ **!問 10.2.2**　G を，四つの頂点 $1, 2, 3, 4$ を持つグラフとする．$1 \leq i < j \leq 4$ に対し，x_{ij} を「頂点 i と頂点 j の間に辺が存在する」ことを意味する命題変数とする．これらの 4 頂点の集合を頂点集合とする任意のグラフは，変数への真理値割り当てによって表現することができる．たとえば，図 10.1 のグラフは，変数 x_{14} を偽にし，残りの五つの変数を真にすることによって表現される．辺が存在するか否かだけに関わるグラフの任意の性質に対し，変数に対する真理値割り当てがその性質を持つグラフを記述するとき，またそのときに限り，真であるようなブール式により，その性質を表現することができる．次の性質を表すブール式を書け．

*a)　G はハミルトン閉路を持つ．

b)　G は連結である．

c)　G はサイズ 3 のクリークを持つ．ただし，サイズ 3 のクリークとは，どの 2 頂点の間にも辺が存在するような 3 頂点を頂点集合とする部分グラフ（すなわち，グラフの中の三角形）である．

d)　G は少なくとも一つ孤立点を持つ．ただし，孤立点とは，その頂点を端点とする辺を一つも持たない頂点である．

10.3 制限された形の充足可能性問題

我々の目標は，10.1.4 項で紹介した TSP 問題などの広い範囲のさまざまな問題に対し，それらが *NP* 完全であることを証明していくことである．そのとき用いる基本的な方針は，充足可能性問題 SAT から対象とする問題への多項式時間還元を見出すことである．ただし，それらの中間に位置する重要な問題で「3SAT」と呼ばれる問題がある．3SAT は，SAT にくらべ，対象とする問題への還元がはるかに容易に構成できることが多い．3SAT も，やはり，ブール式の充足可能性に関する問題であるが，入力される式は非常に均一的な形の式に限られる．3SAT への入力は，ちょうど 3 個の変数，または，否定演算子つきの変数を OR で結んだ「項」のいくつかに対し，それらを AND で結んで得られる式である．

本節では，ブール式に関するいくつかの重要な用語を準備する．次に，任意のブール式に対する充足可能性問題を，入力がある標準形の形の式である充足可能性問題（3SAT 問題）に還元する．任意のブール式 E には，3SAT の標準形の形をしたブール式 F でそれと等価なものが必ず存在するが，F の長さは E の長さに関する指数関数的な大きさにまでなり得る．これは，興味深いことであるが，同時に，注意を払う必要のあることでもある．したがって，SAT から 3SAT への多項式時間還元では，単なるブール代数的な処理とは別の細かい配慮が要求される．SAT のための任意のブール式 E を，3SAT のための標準形の形をした別のブール式 F に変形するが，このとき，F は必ずしも E と等価である必要はない．F が充足可能であるための必要十分条件は E が充足可能であることである，ということさえ確認できればよいのである．

10.3.1 ブール式に対する標準形

基礎的な三つの言葉を定義する．

- 変数，または，否定の演算子を施された変数を**リテラル** (literal) という．
- 一つのリテラル，または，二つ以上のリテラルを論理和 OR で結んだ式を**項** (clause) という．
- ブール式が**乗法標準形** (conjunctive normal form[3], CNF) であるとは，その式がいくつかの項を論理積 AND で結んだ形をしていることである．

[3] "conjunction" とは，論理積 (AND) の別名である．

式を短縮して書くため，もう一つの記法を採用する．すなわち，$\lor$ を和とみなして演算子 $+$ を用い，また，$\land$ を積とみなす．積の表現には，普通，何も書かない．これは，正則表現において連接に対して採用したのと同じ方法である．このような記法を用いると，項を「リテラルの和」と呼び，CNF 式を「項の積」と呼ぶことが自然になる．

例 10.10 ブール式 $(x \lor \neg y) \land (\neg x \lor z)$ を短縮形で書くと $(x+\overline{y})(\overline{x}+z)$ となる．この式は，二つの項 $(x+\overline{y})$ と $(\overline{x}+z)$ の AND（積）の形をしているので，乗法標準形である．

ブール式 $(x+y\overline{z})(x+y+z)(\overline{y}+\overline{z})$ は CNF ではない．この式は，三つの部分式 $(x+y\overline{z})$, $(x+y+z)$, $(\overline{y}+\overline{z})$ の AND の形である．後ろの二つは項であるが，第 1 の部分式は，一つのリテラルと二つのリテラルの積の和であり，項ではない．

ブール式 xyz は CNF である．項は，ただ一つのリテラルだけから成る場合もあることを思い出していただきたい．したがって，この式は三つの項 (x),(y),(z) の積である．　▨

ブール式が **k-乗法標準形** (k-conjunctive normal form, k-CNF) であるとは，その式が，ちょうど k 個の相異なるリテラルの和であるような，いくつかの項の論理積であることである．たとえば，$(x+\overline{y})(y+\overline{z})(z+\overline{x})$ は 2-CNF である．各項がちょうど二つのリテラルを持つからである．

ブール式に関するこれらの制限の各々から，制限に合致するブール式に対してのみ充足可能性を問う，固有の充足可能性問題が生じる．そこで，我々は次のような問題について考えていく．

- CSAT 問題：CNF 形のブール式が与えられたとき，その式は充足可能であるかどうかを判定せよ．
- kSAT 問題：k-CNF 形のブール式が与えられたとき，その式は充足可能であるかどうかを判定せよ．

我々は，CSAT，3SAT，および，3 より大きいすべての k に対する kSAT はいずれも $\mathcal{NP}$ 完全であることを見ていく．一方，1SAT，および，2SAT に対しては線型時間のアルゴリズムが存在する．

悪い入力の取り扱い

これまで考察してきた問題 —— SAT，CSAT，3SAT，等々 —— は，固定された 8 個の記号から成るアルファベット上の言語である．そのアルファベット上の文字列のうちの一部を，我々は，ブール式として解釈する．ブール式として解釈することのできない文字列は，言語 SAT に属すことはない．同様に，制限された形のブール式を考察しているとき，正しい形のブール式ではあっても，制限された形に合致しない式は，考察の対象の言語に属すことはない．したがって，たとえば，CSAT 問題を判定するアルゴリズムは，入力として，充足可能であっても CNF 形ではないブール式が与えられたとき，"no" と答える．

10.3.2 ブール式の CNF 形への変換

二つのブール式が**等価** (equivalent) であるとは，変数に対する任意の真理値割り当てに対し，それらが同じ結果を得ることである．二つのブール式が等価であるならば，明らかに，両者はともに充足可能であるか，それとも，どちらも充足可能でないかのいずれかである．したがって，任意のブール式をそれと等価な CNF 式に変換することは，SAT から CSAT への多項式時間還元の開発に向けて，一つの有力な手段を与える．このような方針をもとにして，CSAT が NP 完全であることを示すことにする．

しかし，事はそれほど単純ではない．確かに，任意のブール式を CNF 形に変換することは常に可能であるのだが，変換には多項式を超える時間が必要となることがあり得る．とくに，変換により式の長さが指数関数的に長くなる場合があり，そのような場合には，明らかに，出力を生成するだけで指数時間を要することになる．

幸いなことに，CSAT が NP 完全であることを証明するためには，任意のブール式の CNF 式への変換は，SAT を CSAT に還元する，という一方向だけでよい．我々がなすべきすべては，SAT の入力例 E を受け取り，それを，F が充足可能であるための必要十分条件は E が充足可能である，という性質を満たす CSAT の入力例 F に変換することだけである．その際，E と F が等価であることは必要ない．さらに，E と F が同じ変数集合を持つことすら必要ではない．実際，以下に示す変換において，F の変数集合は E の変数集合よりも真に大きい場合がある．

SATからCSATへの還元は，二つのステップから構成される．第1のステップでは，否定演算子は変数に直接施される以外には存在しないように，すべての ¬ を「式の木」の下方に押し込んでいく．この結果，ブール式は，いくつかのリテラルにANDとORを施した式になる．この変換により元の式と等価な式が生成される．変換に要する時間は，高々，式の長さに関する2乗の時間である．実際にコンピュータで計算する場合には，注意深く設計されたデータ構造を用いれば，線型時間を要するだけですむ．

第2のステップは，リテラルにANDとORを施した式を，項の積の形に直すこと，すなわち，CNF形に書き換えることである．新しい変数を導入することにより，この変換を，与えられた式の長さに関する多項式の時間で実行することができる．新しく得られる式Fは，一般に，元の式Eと等価ではない．しかし，Fが充足可能であるための必要十分条件はEが充足可能である，という性質は満たされる．もう少し詳しくいうと，TをEを真にする真理値割り当てとするとき，Tの拡張SでFを真にするものが存在する．ただし，SがTの**拡張** (extension) であるとは，Sは，Tが値の割り当てを行う各変数に対してTと同じ値を割り当てるような真理値割り当てである，という意味である．Tが割り当てを行わない残りの変数に対しては，Sはどのような値を割り当ててもよい．

第1のステップは，¬ を ∧ や ∨ の下方（内側）に押し込んでいくことである．そのために必要な規則は次の通りである．

(1) $\neg(E \wedge F) \Rightarrow \neg(E) \vee \neg(F)$．この規則は**ド・モルガンの法則** (De Morgan's laws) の一つであり，¬ を ∧ の下方に押し込むことを可能にする．その副作用として，∧ が ∨ に変わることに注意されたい．

(2) $\neg(E \vee F) \Rightarrow \neg(E) \wedge \neg(F)$．この規則はもう一方の「ド・モルガンの法則」であり，¬ を ∨ の下方に押し込むことを可能にする．副作用として，∨ が ∧ に変わる．

(3) $\neg(\neg(E)) \Rightarrow E$．この規則は**二重否定律** (law of double negation) と呼ばれ，一つの式に施される一対の ¬ を消去する．

例 10.11 ブール式 $E = \neg((\neg(x+y))(\bar{x}+y))$ について考えてみよう．ここでは，否定を表す二つの記法を混在させて用いていることに注意されたい．一つの変数よりも長い式に否定を施すときに演算子 $\neg$ を用いている．図 10.6 に，式 E に対してすべての $\neg$ をリテラルの一部となるまで下方に押し下げていく手順を示す．

ブール式	規則
$\neg((\neg(x+y))(\bar{x}+y))$	元の式
$\neg(\neg(x+y)) + \neg(\bar{x}+y)$	(1)
$x+y+\neg(\bar{x}+y)$	(3)
$x+y+(\neg(\bar{x}))\,\bar{y}$	(2)
$x+y+x\bar{y}$	(3)

図 10.6 否定演算子 $\neg$ を，リテラル以外には出現しないように，「式の木」の下方に押し下げる手順

最後に得られた式は，元の式と等価であり，リテラルの積（AND）の和（OR）の形をしている．実は，この式はさらに単純化することが可能で $x+y$ と等価になる．しかし，この単純化は我々の当面の目標にとって本質的なことではない．我々の目標は，すべての式を演算子 $\neg$ がリテラル以外には出現しない形に書き直すことが可能であることを示すことだからである． ▨

定理 10.12 すべてのブール式 E は，否定演算子がリテラル以外には出現しない形，すなわち，否定演算子は変数に直接施されるだけの形，のブール式 F と等価である．さらに，F の長さは E の記号数に関して線型であり，また，F は E から多項式時間で構成することができる．

証明 証明は，E に含まれる演算子 $(\wedge, \vee, \neg)$ の個数に関する帰納法で行う．我々は，E と等価で，演算子 $\neg$ がリテラル以外には出現しない式 F が存在することを示す．さらに，その際，E が含む演算子の個数を $n\ (\geq 1)$ とするとき，F が含む演算子の個数は $2n-1$ を超えないようにできることも示す．

F について，演算子 1 個につき括弧は 1 対あれば十分であるし，また，式に含まれる変数の個数は演算子の個数より（高々）1 個多いだけである．したがって，F の長さは E の長さの定数倍で抑えられることが結論される．さらに重要なことは，以下に見ていくように，F の構成はとても単純なものなので，F を構成するのに要する時間

はその長さに比例する程度，したがって，E に比例する程度であることである．

基礎：E がただ一つの演算子を持つとき，その形は，変数 x, y に対し，$\neg x$, $x \vee y$, $x \wedge y$ のいずれかである．どの場合も，E はすでに目的の形の式になっている．したがって，$F = E$ でよい．E と F はともに演算子を一個持つので，「F に含まれる演算子の個数は，E に含まれる演算子の個数の 2 倍 マイナス 1 以下である」という関係も成り立つ．

帰納：E より少ない個数の演算子を持つすべてのブール式に対し，証明すべき命題は成り立つと仮定する．まず，E の最も高い位置（最も外側）の演算子が $\neg$ ではない場合を考える．このとき，E は $E_1 \vee E_2$ または $E_1 \wedge E_2$ の形である．どちらの場合も，帰納法の仮定を E_1 と E_2 に適用することができる．すなわち，それぞれに等価な式 F_1 と F_2 で，すべての $\neg$ はリテラルにのみ出現するものが存在する．そこで，$F = F_1 \vee F_2$ または $F = (F_1) \wedge (F_2)$ とすれば，これらが E と等価な目的の式となる．E_1 と E_2 が持つ演算子数を，それぞれ，a, b とする．明らかに，E は $a + b + 1$ 個の演算子を持つ．帰納法の仮定より，F_1 と F_2 は，それぞれ，高々$2a - 1$ 個，または，$2b - 1$ 個の演算子を持つ．したがって，F は高々$2a + 2b - 1$ 個の演算子を持つことになるが，この値は $2(a + b + 1) - 1$，すなわち，E に含まれる演算子数の 2 倍マイナス 1 以下である．

さて次に，E が $\neg E_1$ の形をしている場合を考えよう．このとき，E_1 の最も外側の演算子が何であるかによって，さらに三つの場合に分かれる．E_1 は演算子を持つと仮定してよいことに注意されたい．そうでなければ，E は基礎の場合に帰着されるからである．

(1) $E_1 = \neg E_2$ の場合．二重否定律により，$E = \neg(\neg E_2)$ は E_2 と等価である．E_2 に含まれる演算子の個数は E に含まれる演算子の個数より少ないので，E_2 に対して帰納法の仮定を適用することができる．E_2 と等価で，$\neg$ はリテラルにのみ出現する式 F を見つけることができる．この F が E に対する目的の式となる．F に含まれる演算子の個数は，高々，E_2 に含まれる演算子の個数の 2 倍マイナス 1 であるから，それは，明らかに，E に含まれる演算子の個数の 2 倍マイナス 1 を超えることはない．

(2) $E_1 = E_2 \vee E_3$ の場合．ド・モルガンの法則により，$E = \neg(E_2 \vee E_3)$ は $(\neg(E_2)) \wedge (\neg(E_3))$ と等価である．$\neg(E_2)$ に含まれる演算子の個数も，また，$\neg(E_3)$ に含まれる演算子の個数も E に含まれる演算子の個数より少ないので，帰納法の仮定により，$\neg(E_2)$ と等価であって $\neg$ はリテラルにのみ出現する式 F_2 と，$\neg(E_3)$ と等価であって $\neg$ はリテラルにのみ出現する式 F_3 とが存在する．このとき，$F = (F_2) \wedge (F_3)$ が E と等価な目的の式となる．次に，この F に対し，

アルゴリズムの記述

形式的には，還元のための実行時間は1テープ・テューリング機械上で実行するときに要する時間と定められている．しかし，1テープ・テューリング機械のために設計されるアルゴリズムは不必要に複雑であることが多い．我々は，通常のコンピュータ，多テープTM，1テープTMのいずれを用いる場合でも，多項式の時間で解くことのできる問題の集合は，個々の問題に対する多項式の次数こそ異なるかもしれないが，同じ集合であることを知っている．そこで，ある*NP*完全問題を別の問題に還元するために必要なアルゴリズムが相当に込み入ったアルゴリズムであるような場合，そのアルゴリズムを評価する際，通常のコンピュータ上での効率のよい実現法によって還元に要する計算時間を測定する場合があることを合意しておきたい．そのような共通の理解があれば，テープの操作などに関する瑣末な議論を避けることができ，アルゴリズムの本質的で重要なアイデアに重点を置いて議論を進めていくことが可能となる．

F の中の演算子の個数は過度に多くはないことを示す．E_2 と E_3 とに含まれる演算子の個数を，それぞれ，a, b とする．明らかに，E は $a+b+2$ 個の演算子を持つ．$\neg(E_2)$ と $\neg(E_3)$ とに含まれる演算子の個数は，それぞれ，$a+1, b+1$ である．帰納法の仮定より，F_2 と F_3 とに含まれる演算子の個数は，それぞれ，高々，$2(a+1)-1, 2(b+1)-1$ である．したがって，F に含まれる演算子の個数は，高々，$2a+2b+3$ である．この値は，ちょうど，E に含まれる演算子の個数の2倍マイナス1に等しい．

(3) $E_1 = E_2 \wedge E_3$ の場合．この場合は，ド・モルガンの法則の第2式を用いるが，本質的に (2) の場合と同様である． ▨

10.3.3 CSAT の *NP* 完全性

ここでは，リテラルにANDとORを施して構成されるブール式を受け取り，それをCNF式に変形する問題を考える．前に指摘したように，ブール式 E から，充足可能であるための必要十分条件が E が充足可能であることとなるようなCNF式 F を多項式時間で生成するためには，等価性を保存する還元に固執することをあきらめ，E の中には含まれていない新しい変数を導入して F を構成することが必要になる．このために用いられる「細工」を，CSATは*NP*完全であることを主張する以下の定理の証明の中で紹介する．その後で，その構成法の理解を助けるため，細工の実例を一つ与える．

定理 10.13 CSAT は NP 完全である．

証明 SAT を多項式時間で CSAT に還元する方法を示す．最初に，定理 10.12 の方法を用いて，与えられた SAT の入力例を否定演算子 $\neg$ がリテラル以外には出現しない式 E に変形する．次に，E をある CNF 式 F に多項式時間で変形する方法を示し，F が充足可能であるための必要十分条件は E が充足可能である，ということを示す．F の構成は E の長さに関する帰納法で行う．F が持つ性質の中には，我々が必要とする性質以外のものも含まれる．正確に言うと，E における記号の出現個数（すなわち，長さ $|E|$）に関する帰納法で次のことを示す．

- 長さが n で，$\neg$ がリテラル以外には出現しない任意のブール式 E に対し，次を満たすブール式 F が存在するような，定数 c が存在する．
  - a) F は CNF 式であり，高々n 個の項から成る．
  - b) F は，E から高々$c|E|^2$ の時間で構成することができる．
  - c) E に対する真理値割り当て T が E を真にするための必要十分条件は，T の拡張 S で F を真にするものが存在することである．

基礎：E が一つ，または，二つの記号から成るとする．このとき，E はリテラルである．リテラルは項であり，したがって，E はすでに CNF 式である．

帰納：E より短い任意の式は，項の積の形の式に変形することができること，および，長さ n の式に対するこの変形は高々cn^2 時間で実行できることを仮定する．E の最も上位（外側）の演算子が何であるかにより，二つの場合に分けられる．

場合 1：$E = E_1 \wedge E_2$. 帰納法の仮定より，それぞれ E_1 と E_2 から導かれる CNF 形のブール式 F_1 と F_2 が存在する．E_1 に対する充足可能な真理値割り当てはすべて F_1 に対する充足可能な真理値割り当てに拡張することが可能であり，かつ，その逆も成り立つ．E_2 と F_2 に関しても事情は同じである．一般性を失うことなく，F_1 の変数と F_2 の変数は，E に現れるものを除き，すべて相異なると仮定してよい．すなわち，F_1 や F_2 に新しい変数を導入しなければならないときは，相異なる変数を導入するものとする．

$F = F_1 \wedge F_2$ とおく．F_1 と F_2 が CNF 式ならば，明らかに，$F_1 \wedge F_2$ も CNF 式である．E に対する真理値割り当て T が F を充足する真理値割り当てに拡張可能であるための必要十分条件は，T が E を充足することである，ということを示す．

（十分性）T は E を充足すると仮定する．E_1 に現れる変数の集合への T の制限を T_1 とし，E_2 に現れる変数の集合への T の制限を T_2 する．帰納法の仮定より，T_1 は F_1 を充足する真理値割り当て S_1 に拡張可能であり，T_2 は F_2 を充足する真理値割り当て S_2 に拡張可能である．S_1 の定義域上では S_1 と一致し，S_2 の定義域上では S_2 と

一致するような真理値割り当てを S とする．S_1 の定義域と S_2 の定義域に共通に含まれる変数は E に含まれる変数だけであり，それらの変数に対して S_1 と S_2 の値は一致するので，S は常に定義可能であることに注意されたい．このとき，明らかに，S は T の拡張であり，F を充足する．

（必要性）逆に，T は，F を充足する拡張 S を持つと仮定する．E_1 に現れる変数の集合への T の制限を T_1 とし，E_2 に現れる変数の集合への T の制限を T_2 とする．また，F_1 に現れる変数の集合への S の制限を S_1 とし，F_2 に現れる変数の集合への S の制限を S_2 とする．このとき，S_1 は T_1 の拡張であり，S_2 は T_2 の拡張である．F は F_1 と F_2 を AND で結んだ式なので，S_1 は F_1 を充足しなければならず，S_2 は F_2 を充足しなければならない．そこで，帰納法の仮定より，T_1 は E_1 を充足し，T_2 は E_2 を充足する．したがって，T は E を充足する．

場合 2：$E = E_1 \vee E_2$．場合 1 のように，帰納法の仮定を利用して，次の性質を満たす CNF 式 F_1, F_2 が存在することを主張する．

(1)　E_1 に対する真理値割り当てが E_1 を充足するための必要十分条件は，その真理値割り当てが F_1 を充足する真理値割り当てに拡張可能であることである．同様に，E_2 に対する真理値割り当てが E_2 を充足するための必要十分条件は，その真理値割り当てが F_2 を充足する真理値割り当てに拡張可能であることである．

(2)　F_1 の変数と F_2 の変数は，E に現れるものを除き，すべて相異なると仮定してよい．

(3)　F_1 と F_2 はどちらも CNF 式である．

この場合，単に，F_1 と F_2 を OR で結んだ式を作るだけでは，得られる式は CNF 式にならないので，目的の式 F を得ることはできない．しかし，我々が求めるのは等価性ではなく充足可能性であるという事実を有効に活用して，もう少し複雑な構成を工夫すれば，目的を達することが可能となる．項 g_i, h_j に対し

$$F_1 = g_1 \wedge g_2 \wedge \cdots \wedge g_p$$

および

$$F_2 = h_1 \wedge h_2 \wedge \cdots \wedge h_q$$

であるとする．新しい変数 y を導入し，次のように F を定める．

$$\begin{aligned} F = &(y + g_1) \wedge (y + g_2) \wedge \cdots \wedge (y + g_p) \\ &\wedge(\overline{y} + h_1) \wedge (\overline{y} + h_2) \wedge \cdots \wedge (\overline{y} + h_q) \end{aligned}$$

以下，次のことを証明する．E に対する真理値割り当て T が F を充足する真理値割り当て S に拡張可能であるための必要十分条件は，T が E を充足することである．

（十分性）T は E を充足すると仮定する．場合 (1) のように，E_1 に現れる変数の集合への T の制限を T_1 とし，E_2 に現れる変数の集合への T の制限を T_2 とする．$E = E_1 \vee E_2$ であるので，T は E_1 を充足するか，または，E_2 を充足する．いま，T は E_1 を充足すると仮定しよう．このとき，T の E_1 に現れる変数の集合への制限である T_1 は，F_1 への真理値割り当て S_1 で F_1 を充足するものに拡張することが可能である．T の拡張 S を次のように構成する．その結果，S は上に定義されたブール式 F を充足する．

(1)　F_1 に含まれるすべての変数 x に対し，$S(x) = S_1(x)$ とする．

(2)　$S(y) = 0$ とする．この選択は，F の項で F_2 から導かれるものをすべて真にする．

(3)　F_2 に含まれるが F_1 には含まれない任意の変数 x に対しては，$S(x)$ は 0 または 1 のどちらに定めてもよい．

このとき，規則 (1) により，S は，g_i から導かれる項をすべて真にする．また，y に対する真理値割り当てである規則 (2) により，S は，h_j から導かれる項をすべて真にする．したがって，S は F を充足する．

T が E_1 を充足しない場合は，T は E_2 を充足する．このときも上と同様の論法を適用すればよい．ただし，規則 (2) は $S(y) = 1$ とする．また，$S_2(x)$ が定義されるとき $S(x) = S_2(x)$ とし，F_1 だけに現れる変数に対しては $S(x)$ は任意に定める．この場合も S は F を充足することを結論することができる．

（必要性）E に対する真理値割り当て T は，F に対する真理値割り当て S で，F を充足するものに拡張されると仮定する．y に 0 と 1 のどちらの真理値が割り振られるかにより，二つの場合に分かれる．

まず，$S(y) = 0$ と仮定しよう．このとき，F の項のうち，h_j から導かれる項はすべて真になる．しかし，g_i から導かれる $(y + g_i)$ の形の項に対しては y はまったく無力である．これは，S は g_i 自身をすべて真にしなければならないことを意味する．すなわち，S は F_1 を真にする．

もう少し詳しく述べる．F_1 に現れる変数の集合への S の制限を S_1 とする．S_1 は F_1 を充足する．T を E_1 に現れる変数の集合へ制限したものを T_1 とすると，帰納法の仮定により，T_1 は E_1 を充足しなければならない．その理由は，S_1 は T_1 の拡張であるからである．T_1 は E_1 を充足するので，$E = E_1 \vee E_2$ より，T は E を充足する．

次に，$S(y)=1$ の場合を考える必要がある．しかし，この場合は，すでに上で見た場合と対称的であるので，読者に委ねることにして，ここでは述べない．結論として，S が F を充足するならば，必ず，T は E を充足することがわかる．

E から F を構成する際に要する計算時間は，E の長さ n に関する 2 乗を超えることはないことを示そう．上で分けて考えたどちらの場合でも，E を E_1 と E_2 に分解するための手間と，F_1 と F_2 から F を構成するための手間は，いずれも，E の長さに関する線型の時間である．場合 (1) と場合 (2) のどちらについても，E から E_1 と E_2 を作るために要する時間と F_1 と F_2 から F を作るために要する時間の和に関する一つの上界を dn とする．長さ n の任意のブール式 E から F を構成するために要する時間を $T(n)$ とおく．$T(n)$ について，次のような漸化式が得られる．

$$T(1) = T(2) \le e \qquad \text{ただし}, e \text{は定数である}$$

$$T(n) \le dn + c \max_{0<i<n-1} (T(i) + T(n-i-1)) \qquad n \ge 3 \text{のとき}$$

なお，c は定数であり，あとで，$T(n) \le cn^2$ を成り立たせるように値が定められる．$T(1)=T(2)$ に対する基礎の規則は，E が一つの記号または二つの記号で構成されているならば，E は一つのリテラル以外ではあり得ず，したがって，処理にはある一定の時間 e を要するだけなので，再帰的手続きを用いる必要はない，ということを言っているに過ぎない．再帰的な規則は，次の事実による．E が，二つの部分式 E_1 と E_2 を演算子 $\wedge$ または $\vee$ によって連結した式であるとき，E_1 の長さが i であれば E_2 の長さは $n-i-1$ である．さらに，E から F への変換全体は，高々 dn の時間を要する 2 回の単純な手順 —— E を E_1 と E_2 に変える手順と F_1 と F_2 を F に変える手順 —— ，および，2 回の再帰的な変換 —— E_1 を F_1 に変える変換と E_2 を F_2 に変える変換 —— から構成される．

n に関する帰納法により，次のことを証明する．ある定数 c が存在して，すべての n に対し $T(n) \le cn^2$ が成り立つ．

基礎：$n=1,2$ のとき，e 以上の適当な値を c とすればよい．

帰納：$n \ge 3$ とし，長さが n より短い式に対しては命題が成り立つと仮定する．すなわち，$T(i) \le ci^2$ かつ $T(n-i-1) \le c(n-i-1)^2$ が成り立つ．したがって，

$$T(i) + T(n-i-1) \le c(n^2 - 2i(n-i) - 2(n-i) + 1) \tag{10.1}$$

$n \ge 3$ かつ $0 < i < n-1$ より，$2i(n-i) > n$ かつ $2(n-i) > 2$ が導かれる．したがって，(10.1) 式の右辺は，$0 < i < n-1$ を満たす任意の i に対し，$n^2 - n$ より小さい．よって，$T(n)$ に対する漸化式より $T(n) \le dn + cn^2 - cn$ が得られる．$c \ge d$ を満たすように c を選べば，$T(n) \le cn^2$ が成り立つ．以上で，帰納法の証明が完了

した．

この結果，E から F を構成するために要する時間は $O(n^2)$ であることがわかった．
▨

例 10.14　定理 10.13 の構成を，簡単な例に対し，実際に適用してみよう．例としてとりあげるブール式は $E = x\overline{y} + \overline{x}(y + z)$ である．図 10.7 にこの式の変換木を示す．各頂点の脇に付されているのは，その頂点によって表現される式に対して構成される CNF 式である．

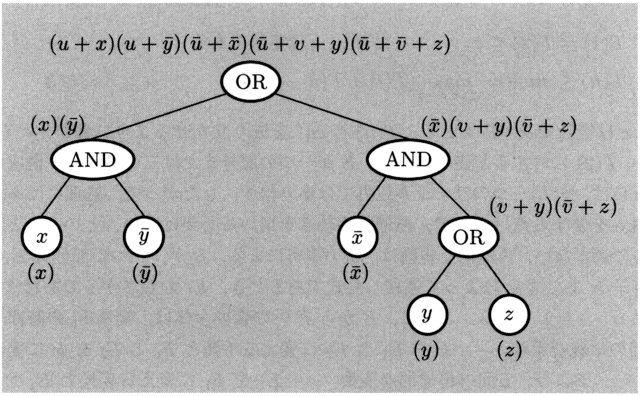


図 10.7　ブール式の CNF 式への変換

葉はリテラルに対応する．各リテラルに対し，構成される CNF 式はそのリテラルだけから成る一つの項である．たとえば，$\overline{y}$ とラベル付けされた葉には CNF 式 $(\overline{y})$ が対応する．括弧は，本来，必要ない．しかし，ここでは，いくつかの項の積について論じているということを強調するため，CNF 式には，敢えて，括弧をつけることにする．

AND の頂点に対し，CNF 式の構成は，二つの部分式に含まれるすべての項の積（AND）をとるだけである．したがって，たとえば，部分式 $\overline{x}(y + z)$ に対する頂点には，$\overline{x}$ に対する一つの項，すなわち $(\overline{x})$，と $y + z$ に対する二つの

項，すなわち $(v+y)$ と $(\overline{v}+z)$，の積である CNF 式が対応する[4]．

OR の頂点に対しては，新しい変数を一つ導入する必要がある．その変数を，左のオペランドに対するすべての項に付加する（和をとる）．また，その変数の否定を，右のオペランドに対するすべての項に付加する．たとえば，図 10.7 の根の頂点を考えてみよう．その頂点は，二つの部分式 $x\overline{y}$ と $\overline{x}(y+z)$ との OR であり，それらの部分式に対応する CNF 式は，それぞれ，$(x)(\overline{y})$ と $(\overline{x})(v+y)(\overline{v}+z)$ である．新しい変数 u を導入し，第 1 のグループの各項には変数をそのまま付加し，第 2 のグループの各項には変数の否定を付加する．結果として，次の式が得られる．

$$F=(u+x)(u+\overline{y})(\overline{u}+\overline{x})(\overline{u}+v+y)(\overline{u}+\overline{v}+z)$$

定理 10.13 は，E を充足する任意の真理値割り当て T は，F を充足する真理値割り当て S に拡張可能であることを主張する．たとえば，割り当て $T(x)=0,\ T(y)=1,\ T(z)=1$ は E を充足する．T を S に拡張するには，T から引き継いだ $S(x)=0,\ S(y)=1,\ S(z)=1$ の他に $S(u)=1,\ S(v)=0$ を付け加えて S を定めればよい．S が F を充足することを確認していただきたい．

S を定める際，u に対しては，$S(u)=1$ とする必要がある．T は E の第 2 の部分式，すなわち $\overline{x}(y+z)$，を真にするだけだからである．E の第 1 の部分式から導かれた二つの項 $(u+x)$ と $(u+\overline{y})$ を真にするためには $S(u)=1$ としなければならない．しかし，v に対しては 0 と 1 のどちらの値を選んでもよい．部分式 $y+z$ では OR の両方のオペランドが T によりすでに真になっているからである．　▨

---

4) この例では，たまたま，部分式 $y+z$ はそのままですでに項になっている．したがって，必ずしも，二つの式の OR に対する一般的な構成法を適用する必要はなく，$y+z$ に等価な項の積として，単に，$(y+z)$ を採用することもできる．しかし，この例の説明では，一般的な規則の適用にこだわりたい．

10.3.4 3SAT の NP 完全性

充足可能性問題が NP 完全であるようなブール式のクラスで，上で考察したクラスよりもさらに小さいクラスを見ていこう．3SAT という問題を復習しておく．

- 各々が3個の相異なるリテラルの和であるような，いくつかの項の積として表されているブール式 E が与えられたとき，E が充足可能であるかどうかを判定せよ．

3-CNF 式の全体は，CNF 式の全体から見るとごくわずかな部分でしかないが，それにも拘わらず，充足可能性の判定が NP 完全となるのに十分なだけの複雑さを持っている．次の定理は，そのことを述べた定理である．

定理 10.15 3SAT は NP 完全である．

証明 SAT が $\mathcal{NP}$ に属すことから，3SAT も明らかに $\mathcal{NP}$ に属す．NP 困難性を証明するため，CSAT から 3SAT への還元を与える．還元の方法は次の通りである．CNF 式 $E = e_1 \wedge e_2 \wedge \cdots \wedge e_k$ が与えられたとき，各項 e_i に対し以下のような置き換えを行い，新しい式 F を作る．F を構成するのに要する時間は E の長さに関する線型の時間である．さらに，次が成り立つことを示す．E に対する真理値割り当てが E を充足するための必要十分条件は，その割り当てが F に対する真理値割り当てで F を充足するものに拡張可能であることである．

(1) 項 e_i が一つのリテラル，たとえば (x)[5]，である場合．二つの新しい変数 u, v を導入し，(x) を四つの項の積 $(x+u+v)(x+u+\overline{v})(x+\overline{u}+v)(x+\overline{u}+\overline{v})$ で置き換える．ここには u と v の4通りの組合せがすべて出現しているので，この積を充足する唯一の手段は x を真にすることである．したがって，E を充足する任意の真理値割り当ては F を充足する真理値割り当てに拡張可能であるし，また，その逆も成り立つ．

(2) 項 e_i が二つのリテラルの和 $(x+y)$ である場合．一つの新しい変数 z を導入し，e_i を二つの項の積 $(x+y+z)(x+y+\overline{z})$ で置き換える．場合 (1) と同様，両方の項を充足するための唯一の方法は $(x+y)$ を充足することである．

(3) 項 e_i が三つのリテラルの和である場合．すでに 3-CNF として要請される形をしているので，新たに構成される式 F の中に e_i をそのまま残す．

---

5) 面倒を省くため，ここではリテラルを，x のように，否定演算子が施されていない変数として扱っている．しかし，いくつかまたはすべてのリテラルが，$\overline{x}$ のように，否定演算子付きの変数である場合でも，ここに述べる構成は同様にうまく機能する．

(4)　ある $m \geq 4$ に対し $e_i = (x_1 + x_2 + \cdots + x_m)$ である場合．新しい変数 $y_1, y_2, \ldots, y_{m-3}$ を導入し，e_i を次の $m-2$ 個の項の積で置き換える．

$$(x_1 + x_2 + y_1)(x_3 + \overline{y_1} + y_2)(x_4 + \overline{y_2} + y_3)\cdots$$
$$\cdots(x_{m-2} + \overline{y_{m-4}} + y_{m-3})(x_{m-1} + x_m + \overline{y_{m-3}}) \tag{10.2}$$

E を充足する真理値割り当て T は，e_i の中の少なくとも一つのリテラルを真にしなければならない．いま，x_j を真にすると仮定する．(x_j は変数，または，変数の否定であることに注意.) このとき，$y_1, y_2, \ldots, y_{j-1}$ をすべて偽にし，$y_j, y_{j+1}, \ldots, y_{m-3}$ をすべて真にすれば，(10.2) に現れるすべての項が充足されることがわかる．したがって，T を，これらの項のすべてを充足するように拡張することができる．一方，T がすべての x_i を偽にするとき，(10.2) を真にするように T を拡張することは不可能である．理由は，全部で $m-2$ 個の項があるが，一方，$m-3$ 個の y_j の各々は，それが真であるか偽であるかによらず，ただ一つの項しか真にすることができないからである．

以上により，CSAT の任意の入力例 E を 3SAT の入力例 F に，F が充足可能であるための必要十分条件は E が充足可能である，という性質を満たすように変換する方法を示すことができた．この変換に要する時間は，明らかに，E の長さに関する線型の時間である．何故ならば，上の四つの場合のいずれについても，与えられた項を 32/3 倍（これは，場合 (1) における記号の個数の比）より長く引き伸ばすことはなく，また，F の構成に必要な記号を，記号の個数に比例する時間で計算することは容易だからである．CSAT が *NP* 完全であることから，3SAT も同様に *NP* 完全であることが導かれる．　▨

10.3.5　10.3 節の演習問題

♣　**問 10.3.1**　次のブール式を 3-CNF 式に直せ．

*a)　$xy + \overline{x}z$

b)　$wxyz + u + v$

c)　$wxy + \overline{x}uv$

♣　**問 10.3.2**　**4TA-SAT 問題**を次のように定める．ブール式 E が与えられたとき，E に対する真理値割り当てで E を充足するものが少なくとも 4 個存在するかどうかを判定せよ．4TA-SAT は *NP* 完全であることを示せ．

♣　**問 10.3.3**　3-CNF 式から成る一つの族 $\{E_3, E_4, \ldots\}$ を次のように定義する．式 E_n は n 個の変数 $x_1, x_2, \ldots, x_n$ を持つ．E_n は，$\{1, 2, \ldots, n\}$ に属する相異なる 3 個の整数の組 (i, j, k) のすべてに対する $(x_i + x_j + x_k)$ と $(\overline{x_i} + \overline{x_j} + \overline{x_k})$ の積である．以下の n に対し，E_n は充足可能であるかどうかを判定せよ．

***!**a) $n = 4$

!!b) $n = 5$

♣ **! 問 10.3.4** 2SAT 問題を解く多項式時間のアルゴリズムを求めよ．ただし，2SAT 問題とは，各項が二つのリテラルだけから成る CNF 式に対する充足可能性問題である．

ヒント：各項において，二つのリテラルのうちの一方が偽であれば，もう一方は，必然的に，真であることが要請される．一つの変数の真理値をどちらかに仮定することから始め，他の変数に対する必然の帰結をすべて追いかけてみよ．

10.4 その他の *NP* 完全問題

一つの *NP* 完全問題から別の問題の *NP* 完全性の証明を導く方法について，サンプルをいくつか紹介する．新しい *NP* 完全問題を発見するためのこの方法には，二つの重要な効果が伴う．

- ある問題について，その問題が *NP* 完全であることが証明されたとき，そのことが我々に教えてくれるのは，その問題を解くための効率的なアルゴリズムの開発はほとんど望みがない，ということである．我々には，その場合，発見的方法，部分解，近似解，あるいは，その他の手段で，問題を正面からまともに攻撃することを避けるための方策を探すことが奨励される．そして，そのとき我々は，「上手な工夫を見逃した」結果としてそうしているのではなく，自信を持ってそうしているのだ，ということができるのである．
- *NP* 完全問題のリストに新しい問題 P を付け加えるたびに，**すべての** *NP* 完全問題は指数時間を必要とするという見方を，改めて，強く認識することとなる．問題 P に対し多項式時間のアルゴリズムを見つけることを目指してなされた努力は，そのように認識されてはいなかったかもしれないが，$\mathcal{P} = \mathcal{NP}$ を示すために捧げられた努力であった．多くの有能な科学者や数学者によってなされてきた，$\mathcal{P} = \mathcal{NP}$ と同等な何らかの事柄を示すための試みで不成功に終わったものの蓄積こそが，我々に $\mathcal{P} = \mathcal{NP}$ が成り立つことはほとんどあり得ないこと，そして，**すべての** *NP* 完全問題は指数時間を必要とするであろうことを，究極的に確信させる要因となるのである．

本節では，グラフにまつわるいくつかの *NP* 完全問題を考察していく．以下で取り上げる問題は，グラフに関連する問題のうちの代表的なものであり，また，実用的な重要性を持つ問題としてもっともよく現れるもののうちのいくつかでもある．すでに 10.1.4 項で見た巡回セールスマン問題（TSP）も，改めて，取り上げる．それよりもやさしく，しかし同様に重要な問題であるハミルトン閉路問題（HC）が *NP* 完全であることを示し，そのことから，それより一般的な問題である TSP の *NP* 完全性を導く．その他に，グラフの「被覆」に関する問題も紹介していく．任意の辺に対しその一方または両方の端点を含むという意味で，すべての辺を「被覆」する頂点の集合を考え，その中で極小のものを求めることを要求する「頂点被覆問題」がその一例である．

10.4.1 *NP* 完全問題の記述

新しい *NP* 完全問題を導入する際，定義を記述するために，次のような統一的な形式を採用することにする．

(1)　問題の**名前**，および，(多くの場合）略記法．例：3SAT，TSP
(2)　問題に対する**入力**．　何を，どのように表現して，入力するか．
(3)　要求される**出力**．　どのような状況のもとで "yes" を出力するか．
(4)　*NP* 完全性を証明する際，還元のもととして利用される問題

例 10.16　3SAT 問題の定義と *NP* 完全性の証明に使われる問題の記述の例を示す．

問題：3-CNF 式に対する充足可能性問題（3SAT）
入力：3-CNF 形のブール式
出力：式が充足可能であるとき，またそのときに限り，"yes" を出力
還元のもととなる問題：CSAT　▨

10.4.2 独立集合問題

G を無向グラフとする．G の頂点集合の部分集合 I に対し，I に属すどの 2 点も G の辺で結ばれることがないとき，I は**独立集合** (independent set) であるという．独立集合は，同じグラフに対する独立集合の中で最大である（最大個数の頂点を持つ）とき，**極大** (maximal) であるという．

例 10.17 図 10.1 のグラフ（10.1.2 項参照）では，$\{1,4\}$ は極大独立集合である．$(1,4)$ 以外の任意の頂点対に対しては必ず辺が存在するので，この集合は，大きさ（要素数）が 2 の集合の中で独立なただ一つの集合である．また，大きさが 3 以上の集合はどれも独立ではない．たとえば，頂点 1 と 2 の間には辺があるので，$\{1,2,4\}$ は独立ではない．したがって，$\{1,4\}$ は極大独立集合である．実際，この集合は，このグラフに対するただ一つの極大独立集合である．ただし，一般には，一つのグラフに対し多数の極大独立集合が存在することがあり得る．また，$\{1\}$ はこのグラフに対する独立集合であるが，極大ではない．
▨

組合せ論的最適化の方面では，極大独立集合問題は，通常，次のように記述される．「与えられたグラフに対し，極大独立集合を求めよ．」しかし，実行不能問題の理論において扱われるすべての問題と同様，ここでは，yes/no を答える判定問題として問題を記述しなければならない．そのため，問題の定義の中に限界値（下界）を導入する必要が生じる．そして，問題を次のような形で記述する．与えられたグラフが，与えられた下界以上の大きさの独立集合を持つかどうかを判定せよ．極大独立集合問題の形式的な定義を次に与える．

問題：独立集合問題（IS）

入力：グラフ G と下界 k．ただし，k は 1 以上で G の頂点数以下の整数．

出力：G が頂点数 k の独立集合を持つとき，またそのときに限り，“yes” を出力

還元のもととなる問題：3SAT

IS が NP 完全であることを，上での約束通り，3SAT からの多項式時間還元を与えることによって証明する．この還元は次の定理の証明の中で与えられる．

定理 10.18 独立集合問題は NP 完全である．

証明 まず，IS が $\mathcal{NP}$ に属すことを見るのはやさしい．グラフ G と下界 k が与えられたとき，k 個の頂点を推測，選択し，それらが独立であることを確かめればよい．

次に，3SAT から IS への変換の還元の方法を示そう．$E = (e_1)(e_2)\cdots(e_m)$ を 3-CNF 式とする．E から $3m$ 個の頂点を持つグラフ G を構成する．頂点には，$[i,j](1 \le i \le m,\ j = 1,2,3)$ という名前をつける．頂点 $[i,j]$ は，項 e_i の中の j 番目のリテラルに対応する．図 10.8 は，3-CNF 式

$$(x_1 + x_2 + x_3)(\overline{x_1} + x_2 + x_4)(\overline{x_2} + x_3 + x_5)(\overline{x_3} + \overline{x_4} + \overline{x_5})$$

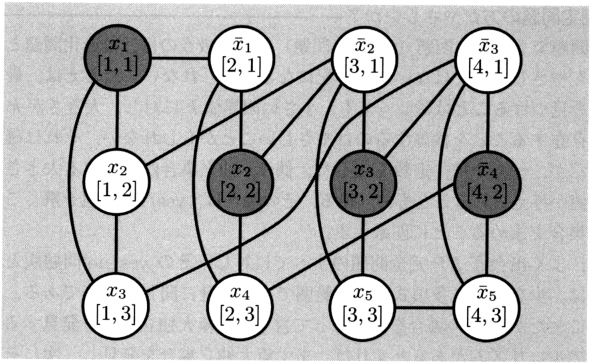


図 10.8　充足可能な 3-CNF 式からの独立集合の構成

に対するグラフの例である．一つの列が一つの項を表す．辺の定め方については，すぐ後で説明する．

G の構成の背後にある「工夫」は，辺の集合に次のような意味を持たせることである．m 個の頂点から成る任意の独立集合は，ブール式 E を充足するための方法を表現する．そのための，鍵になるアイデアは二つある．

(1)　一つの項ごとに頂点がただ一つだけ選ばれることを確実にしたい．このことを実現するため，各列において任意の 2 頂点を辺で結ぶことにする．すなわち，図 10.8 のように，すべての i に対し，三つの辺 $([i,1],[i,2]),([i,1],[i,3]),([i,2],[i,3])$ を作る．

(2)　独立集合を構成する頂点として，ある変数とその変数に否定演算子をつけたリテラルとがともに選ばれる，という事態は避けねばならない．そこで，一方が変数 x を表し，他方が $\overline{x}$ を表すような二つの頂点があるとき，これらの 2 頂点を辺で結ぶことにする．そうすることにより，一つの独立集合の中にこのような頂点の両方が選ばれることは不可能になる．

これらの二つの規則によって構成されるグラフ G に対し，下界 k としては m をとればよい．

ブール式 E からグラフ G と下界 k を構成する作業は，E の長さの 2 乗程度の時間で行うことができるが，そのことを見るのは難しくない．したがって，E から G への変形は多項式時間還元である．次に，この還元が 3SAT から IS への正しい還元であることを示そう．すなわち，次のことを証明する．

● E が充足可能であるための必要十分条件は，G が大きさ m の独立集合を持つこ

判定問題の方がやさしいか？

同じ問題でも，判定問題（yes/no 問題）として扱うのは，最適化問題として扱うよりもやさしいのではないかと心配になるかもしれない．たとえば，最大の独立集合を見つけることは難しくても，小さい限界値 k に対し，大きさが k の独立集合が存在することを検証するのはやさしいことかもしれない．それは確かにその通りだが，一方では，定数 k として，偶々，独立集合が存在する大きさとして最大の値が与えられることもあり得る．その場合は，yes/no 問題を解くことは極大独立集合を求めることに匹敵する．

実際，よく出会う NP 完全問題のすべてに対し，その yes/no 問題版と最適化問題版は，少なくとも多項式の差の範囲で，計算量に関して同等である．一般に言えることだが，IS の場合を例にとって言うと，極大独立集合を**発見**する多項式時間のアルゴリズムがあるとすれば，まず極大独立集合を発見し，次にその大きさが限界値 k 以上であるかどうかを見ることにより，yes/no 問題を解くことができる．yes/no 問題は NP 完全であることが証明されるので，最適化問題も同様に実行不能であるに違いないことがわかる．

逆の方向の比較も可能である．yes/no 問題 IS に対する多項式時間のアルゴリズムが存在したと仮定する．与えられたグラフの頂点数が n であるとき，極大独立集合の大きさは 1 以上，n 以下である．そこで，1 から n までのすべての値に対し，その値を限界値とする yes/no 問題 IS を解くことにより，IS を 1 回解くのに要する時間の n 倍程度の時間で，確かに，（極大独立集合そのものではないかもしれないが）極大独立集合の大きさを求めることができる．さらに付言すると，2 分探索を用いれば，1 回の実行時間の $\log_2 n$ 倍程度の時間しか必要としないこともわかる．

とである．

（十分性）まず，任意の独立集合は，同じ一つの項に属す二つの頂点，すなわち，ある $j_1 \neq j_2$ に対する $[i, j_1]$ と $[i, j_2]$，の両方を含むことはないことに注意する．その理由は，図 10.8 の各列を見てもわかるように，そのような頂点の任意の対に対して必ず辺が存在するからである．したがって，大きさが m の独立集合があるとき，この集合は m 個の項の各々からちょうど一つの頂点を含む集合でなければならない．

さらに，独立集合は，変数 x とその否定 $\overline{x}$ の各々に対応する頂点の両方を含むことはない．その理由は，そのような頂点のすべての対に対しても，その 2 頂点を結ぶ辺が必ず存在するからである．これらのことから，大きさが m の独立集合 I に対し，E を充足する真理値割り当て T を次のようにして構成することができる．変数 x に対応

する頂点が I に属すとき，$T(x)=1$ と定める．否定演算子が施されたリテラル $\overline{x}$ に対応する頂点が I に属すとき，$T(x)=0$ と定める．x に対応する頂点も，また，$\overline{x}$ に対応する頂点も I に属さないときは，$T(x)$ の値は任意でよい．上記の項目 (2) が，x に対応する頂点と $\overline{x}$ に対応する頂点がともに I に属してしまい矛盾が生じるという事態の起こり得ない理由を説明していることに注意されたい．

このとき，T は E を充足する．その理由は，E の各項は，必ず，I に属すリテラルの一つに対応する頂点を持ち，また，T はそのようなリテラルの値を真にするように定められているからである．よって，大きさが m の独立集合が存在するならば，E は充足可能である．

(必要性) E は，ある真理値割り当て T によって充足されると仮定する．T は E のすべての項を真にするので，各項から一つずつ T が真にするリテラルを選ぶことができる．いくつかの項については，二つまたは三つのリテラルが選択可能であるかもしれない．その場合は，そのうちの一つを任意に選ぶ．各項から選ばれたリテラルに対応する頂点をすべて取り出すことにより，m 個の頂点から成る集合 I を構成する．

このとき，I は独立集合である．各項から頂点をただ一つだけ取り出しているので，同じ項（図 10.8 の各列）から得られた 2 頂点を結ぶ辺に対し，その両端の頂点がともに I に属すということはあり得ない．また，真理値割り当て T により真とされたリテラルに対応する頂点だけを I の要素として選んでいるので，一つの変数とその否定を結ぶ辺に対し，その両端の頂点がともに I に属すということもあり得ない．T は x と $\overline{x}$ のうちの一つを真にするが，両方を真にすることはないからである．結論として，E が充足可能であるとき，G は大きさが m の独立集合を持つことが導かれる．

以上より，3SAT から IS への多項式時間還元が存在することがわかった．3SAT は *NP* 完全であることがすでにわかっているので，定理 10.5 により IS も *NP* 完全である．▨

例 10.19　定理 10.18 の構成がどのように行われるかを，次式の場合を例にとって見てみよう．

$$E=(x_1+x_2+x_3)(\overline{x_1}+x_2+x_4)(\overline{x_2}+x_3+x_5)(\overline{x_3}+\overline{x_4}+\overline{x_5})$$

このブール式から得られるグラフは，すでに図 10.8 に示されている．頂点の全体は，四つの項の各々に対応する四つの列に分かれる．各頂点に対し，その名前（整数の対）だけでなく，それが対応するリテラルも記した．一つの列に属す頂点は一つの項に属すリテラルに対応する．一つの列に属す 2 頂点の対のすべてに対し，その 2 頂点を結ぶ辺がある．また，各変数に対応する頂点とその否定に対応する頂点の間にも，必ず，辺がある．たとえば，$\overline{x_2}$ に対応する頂点 [3, 1] は，それぞれ x_2 の出現に対応する二つの頂点 [1, 2] と [2, 2] への辺を持つ．

独立集合は何の役に立つのか？

NP 完全であることを証明する問題の応用について解説することは，本書の目的ではない．しかし，10.4 節で取り上げる問題はカープの NP 完全性に関する基本的な論文の中から選択されたものであり，その論文は，オペレーションズ・リサーチ（OR）の分野からいくつかの最も重要な問題を選んで調べ，それらのうちの多くが NP 完全であることを示したものである．これらの「抽象的な」問題を利用して解くことができる問題が数多く存在することについて,「現実」の問題から得られる豊富な証拠がある．

例を一つ挙げよう．大きい独立集合を見つけるための効率のよいアルゴリズムを，定期試験の時間割の作成に役立てることができる．グラフの各頂点は授業の各々のコマに対応するものとする．二つの頂点の間を辺で結ぶのは，対応する授業を両方とも履修する学生が少なくとも 1 人いて，したがって，その二つの授業のための試験を同じ時間に実施することはできないときであるとする．このグラフに対して極大独立集合を求めれば，その集合に属すすべての授業の試験は同じ時間に実施してもよいことがわかる．このとき，どの学生についても二つの科目の試験時間が重複することはない．

我々は，各列から一つずつ，全部で四つの頂点を選び，集合 I を作った．選ばれた頂点は太線で囲んで示した．この集合は，明らかに，独立集合である．これらの 4 頂点に対応するリテラルは $x_1, x_2, x_3, \overline{x_4}$ であり，我々は，これらから真理値割り当て T を $T(x_1) = 1$，$T(x_2) = 1$，$T(x_3) = 1$，$T(x_4) = 0$ として定めることができる．x_5 に対する真理値割り当ても必要であるが，これは任意に定めればよく，たとえば，$T(x_5) = 0$ とすればよい．こうすれば，T は E を充足する．頂点の集合 I は，T によって真となるように，各項から選ばれたリテラルを示す． ▨

10.4.3 頂点被覆問題

組合せ論的な最適化問題のクラスでもう一つの重要なクラスとして，グラフの「被覆」にまつわる問題のクラスがある．たとえば，**辺被覆** (edge cover) は，辺の集合であって，グラフの中の任意の頂点がその集合に属す少なくとも一つの辺の端点となっているようなものである．辺被覆が**極小** (minimal) であるとは，そのグラフに対するすべての辺被覆の中で辺の個数が最小であることである．与えられたグラフが k 個の辺からなる辺被覆を持つかどうかを判定する問題は NP 完全である．ただし，ここでは，そのことの証明は記さない．

一方，**頂点被覆** (node cover) の問題も NP 完全であるが，これについては証明を紹介する．グラフの頂点被覆とは，頂点の集合であって，任意の辺に対し少なくとも一つの端点がその集合に属すようなものである．頂点被覆が**極小** (minimal) であるとは，そのグラフに対するすべての頂点被覆の中で頂点の個数が最小であることである．

頂点被覆と独立集合は，相互に密接な関係を持つ．実際，独立集合の「補集合」は頂点被覆であり，頂点被覆の「補集合」は独立集合である．したがって，頂点被覆問題（NC）を yes/no 問題としてきちんと記述すれば，IS からの還元を構成することは極めて容易である．

問題：頂点被覆問題（NC）

入力：グラフ G と上界 k．ただし，k は 1 以上で G の頂点数未満の整数．

出力：G が頂点数 k 以下の頂点被覆を持つとき，またそのときに限り，“yes” を出力

還元のもととなる問題：独立集合問題

定理 10.20　頂点被覆問題は NP 完全である．

証明　明らかに，NC は $\mathcal{NP}$ に属す．k 個の頂点の集合を推測し，G のすべての辺についてその端点のうち少なくとも一方はその集合に属すことを確かめればよい．

証明を完成するため，IS から NC への還元を示す．還元のアイデアは，図 10.8 から示唆されるように，独立集合の補集合は頂点被覆であるというところにある．たとえば，図 10.8 のグラフにおいて，太線で囲まれていない頂点全体の集合は頂点被覆を構成する．実は，この場合，太線で囲まれている頂点の集合は極大独立集合であるので，それ以外の頂点の集合は極小頂点被覆となっている．

還元は次のように行えばよい．グラフ G と下界 k を独立集合問題への入力例とする．G の頂点数が n であるとき，G と上界 $n-k$ を頂点被覆問題への入力例とする．この変形は，明らかに，線型時間で実行することができる．次のことを示す．

- G が大きさ k の独立集合を持つための必要十分条件は，G が大きさ $n-k$ の頂点被覆を持つことである．

（十分性）G の頂点の集合を N とし，大きさが $n-k$ の頂点被覆を C とする．このとき，$N-C$ は独立集合であることを示す．そのため，いま，$N-C$ は独立集合ではないと仮定する．すなわち，$N-C$ に属す二つの頂点 v と w で，G において辺で結ばれているものがあるとする．ところが，このとき，v も w も C に属さないので，

G の中の辺 $\{v, w\}$ は頂点被覆であるはずの C によって被覆されないことになってしまう．矛盾が生じたので，$N - C$ は独立集合であることが示された．明らかに，この集合は k 個の頂点を持つ．以上より，十分条件の証明が完了した．

(必要性) k 個の頂点から成る独立集合を I とする．このとき，$N - I$ は $n - k$ 個の頂点を持つ頂点被覆であることを示す．再び，背理法を用いる．$N - I$ によって被覆されない辺 $\{v, w\}$ が存在するとする．このとき，v と w はともに I に属すが，両者は辺で結ばれている．これは，独立集合の定義に矛盾する． ▨

10.4.4 有向ハミルトン閉路問題

これから先は，巡回セールスマン問題（TSP）の NP 完全性を示すことを目標とする．我々が TSP 問題を重視するのは，この問題が組合せ論の分野で最も関心を引く問題の一つだからである．TSP 関連の問題の NP 完全性の証明で最もよく知られているのは，実は，もう少し単純な問題で「ハミルトン閉路問題」(HC) と呼ばれる問題の NP 完全性の証明である．**ハミルトン閉路問題** (Hamilton-Circuit Problem) は次のように記述される．

問題：ハミルトン閉路問題（HC）

入力：無向グラフ G.

出力：G が**ハミルトン閉路** (Hamilton circuit) を持つとき，すなわち，G のすべての頂点をちょうど 1 回通る閉路が存在するとき，またそのときに限り，“yes” を出力．

HC 問題は TSP の特別の場合，すなわち，辺の重みがすべて 1 の場合であることに注意されたい．したがって，HC から TSP への多項式時間還元は極めて容易である．与えられたグラフの各辺に重み 1 を付加するだけでよい．

HC に対する NP 完全性の証明はかなり難しい．我々がとる方法は，HC そのものについて考察する前に，HC にさらに制約をつけた問題を導入することである．制約とは，辺は方向を持ち（すなわち，辺は有向辺である），ハミルトン閉路として考える閉路は辺を正しい向きにたどって得られる閉路に限る，というものである．我々は，まず，3SAT をこの有向グラフに対する HC 問題に還元する．そして，その後で，有向グラフに対する HC 問題を無向グラフに対する，いわば標準的な，HC に還元する．有向グラフに対する問題を形式的に記述してみよう．

問題：有向ハミルトン閉路問題（DHC）

入力：有向グラフ G.

出力：G の中にすべての頂点をちょうど 1 回通る有向閉路が存在するとき，またそのときに限り，“yes” を出力

還元のもととなる問題：3SAT

定理 10.21　有向ハミルトン閉路問題は NP 完全である.

証明　DHC が $\mathcal{NP}$ に属すことの証明はやさしい．有向閉路を推測，選択し，その中にすべての頂点がちょうど 1 回ずつ含まれていることを確かめればよい．

次に，3SAT を DHC へ還元する．この還元では，複雑なグラフの構成が要求される．3SAT への入力例の中の各変数や各項に対応して用意されるある種の「仕掛け」(特殊な部分グラフ）を含むグラフがそれである．

3-CNF 形のブール式から DHC の入力を構成していく．与えられたブール式を $E = e_1 \wedge e_2 \wedge \cdots \wedge e_k$ とする．ただし，各 e_i は 3 個のリテラルの和である項 $e_i = (\alpha_{i1} + \alpha_{i2} + \alpha_{i3})$ である．E に含まれる変数の全体を $x_1, x_2, \ldots, x_n$ とする．各項と各変数に対し，図 10.9 に示すような「仕掛け」を構成する．

各変数 x_i に対し，図 10.9(a) に示された構造を持つ部分グラフ H_i を構成する．ここで，m_i は E における x_i の出現数と $\overline{x_i}$ の出現数のうちの大きい方の値とする．二つの頂点の列 b と c に対し，b_{ij} と c_{ij} の間で両方向に有向辺がある．また，b の各頂点から一つ下の段の c へ向けて有向辺がある．正確に言うと，$j < m_i$ のとき，b_{ij} から $c_{i,j+1}$ への有向辺がある．同様に，$j < m_i$ のとき，c_{ij} から $b_{i,j+1}$ への有向辺がある．最後に，先頭の頂点 a_i は b_{i0} と c_{i0} への有向辺を持ち，また，末尾の頂点 d_i は b_{im_i} と c_{im_i} からの有向辺を持つ．

図 10.9(b) にグラフ全体の構造の概略を示す．一つの六角形は，一つの変数に対する図 10.9(a) の構造をもつ仕掛けを表す．一つの仕掛けの末尾の頂点から，環状に，次の仕掛けの先頭の頂点への有向辺がある．

図 10.9(b) のグラフに対し，有向ハミルトン閉路が存在すると仮定する．さらに，閉路の出発点を a_1 と仮定しても差し支えない．このとき，仮りに，a_1 の次に b_{10} に行くとすると，その次には c_{10} に行かなければならない．そうでないと，c_{10} が閉路上に出現することができなくなるからである．もう少しきちんと言うと，閉路が a_1, b_{10}, c_{11} の順に進むとすれば，c_{10} を終点とする有向辺の二つの始点（すなわち，a_1 と b_{10}）がすでに閉路上に出現してしまったため，閉路が c_{10} を含むことは不可能になるからである．

したがって，閉路が a_1,b_{10} の順に進んだとすると，その後,「梯子」を左右に行き来

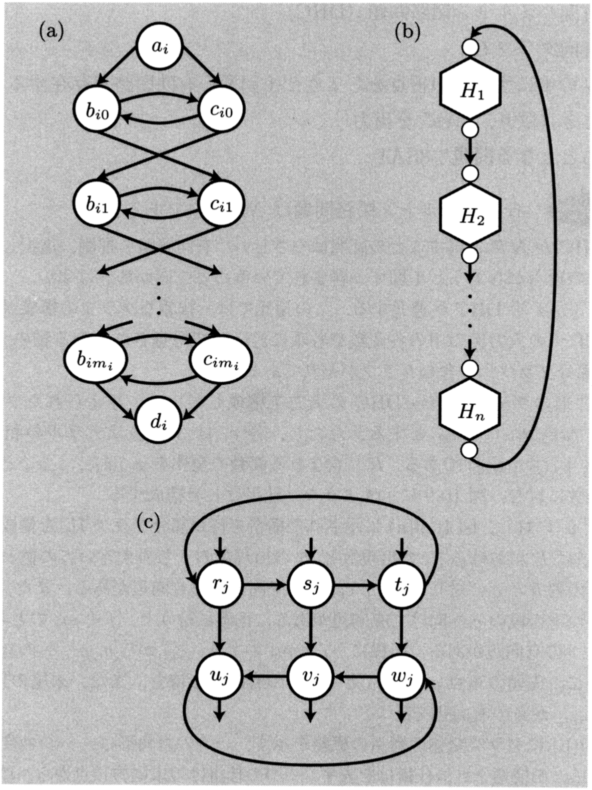


図 10.9　有向ハミルトン閉路問題に対する NP 完全性の証明の中で用いられる構成

しながら下り続けることが必要になる．すなわち，次の順に進む．

$$a_1, b_{10}, c_{10}, b_{11}, c_{11}, \ldots, b_{1m_1}, c_{1m_1}, d_1$$

一方，閉路が a_1,c_{10} の順に進むとすれば，同じ段では c が b より先になるような順序で梯子を降りていくことになる．すなわち，次の順である．

$$a_1, c_{10}, b_{10}, c_{11}, b_{11}, \ldots, c_{1m_1}, b_{1m_1}, d_1$$

証明の中で肝要な点は，2 通りの進み方のうち，c から下段の b に進む第 1 の順序は，

その仕掛けに対応する変数の値を真にすることを意味すると解釈し，一方，b から下段の c に進む第 2 の順序は，その変数の値を偽にすることを意味すると解釈するところにある．

仕掛け H_1 を通過したのち，閉路は a_2 に進む．ここで再び，次に b_{20} に行くかまたは c_{20} に行くかの選択がある．H_1 について論じたのと同様，a_2 から左に進むか，あるいは，右に進むかの選択をすれば，そのあとの H_2 内部での進み方は一意的に定まる．一般に，各 H_i に入るたびに，先頭の頂点の次に左に進むのか，あるいは，右に進むのかの選択があり，そこで選択をしたのちは，**到達不能** (inaccessible) の頂点（その頂点を終点とする有向辺の始点がすでにすべて使われてしまったため，有向閉路上に出現することのできない頂点）を残さないためには，そのあとの進み方は一意的に定まる．

以下の議論では，a_i から b_{i0} へ進む路を選択することは変数 x_i の値を真にすることに対応し，一方，a_i から c_{i0} へ進む路を選択することは変数 x_i の値を偽にすることに対応すると思うとよい．したがって，図 10.9(b) のグラフは，n 個の変数に対する 2^n 個の真理値割り当てに対応して，ちょうど 2^n 個の有向ハミルトン閉路を持つ．

ただし，図 10.9(b) は，まだ，3-CNF 式 E に対して生成するグラフの骨組みに過ぎない．各項 e_j に対して，図 10.9(c) に示すようなもう一つの部分グラフ I_j を導入する．仕掛け I_j は次の性質を持つ．ハミルトン閉路が仕掛け I_j に入る場合，r_j から入れば必ず u_j から抜け出し，s_j から入れば必ず v_j から抜け出し，さらに，t_j から入れば必ず w_j から抜け出す．我々が示したいのは，次のことである．閉路が I_j に到達したとき，入ってきた頂点の真下にある頂点から抜け出すことをしないと，一つまたは二つ以上の頂点が到達不能，すなわち，閉路上に出現することができないことになる．対称性により，r_j が，閉路上に現れる I_j の最初の頂点である場合だけを考えれば十分である．

(1) 閉路上の次の二つの頂点が s_j と t_j である場合．閉路が次に w_j に行き，そこから I_j を抜け出すとすると，v_j は到達不能になる．また，閉路が次に w_j, v_j と進み，そこから I_j を抜け出すとすると，u_j は到達不能になる．したがって，閉路は，仕掛けの中の 6 個の頂点をすべて通過した後，u_j から抜け出すしかない．

(2) r_j の次の二つの頂点が s_j と v_j である場合．閉路が次に u_j に行かないとすると，u_j は到達不能になる．u_j に行き，その直後 w_j に行くとすると t_j が閉路の中に現れることはできない．それを示すための議論は，到達不能性の議論を「逆向き」にすればよい．いまの場合，外部から t_j に到達することはあり得る．しかし，閉路が後に t_j に到達したとすると，次に進むことの可能な頂点は残されていない．t_j の次に進む可能性のある頂点は両方ともすでに閉路の前の部分で使われてしまっているからである．したがって，この場合もまた閉路は u_j から

抜け出すことになる．このとき，t_j と w_j は通過されずに残ってしまうことに注意されたい．それらは，さらに後の部分で閉路に取り込まれる必要がある．そしてそれは可能なことである．

(3) 閉路が r_j から直接 u_j に進む場合．その次に閉路が w_j に進むとすると t_j を閉路の中に含めることはできなくなる．(2) の場合の議論と同様，t_j の次に進む可能性のある頂点は両方ともすでに使われてしまっているからである．したがって，この場合，閉路は u_j から直接外部に抜け出さなければならない．通過せずに終わった残りの 4 個の頂点は後に改めて訪れることになる．

ブール式 E に対するグラフ G の構成を完了するため，次の規則にしたがって I_j を H_i に接続する．項 e_j の最初のリテラルは否定演算子のついてない変数 x_i であるとする．仕掛け I の一つに接続する目的ではまだ使われていない一つの頂点 c_{ip} $(0 \leq p \leq m_i - 1)$ を選び，c_{ip} から r_j へ向かう有向辺と u_j から $b_{i,p+1}$ へ向かう有向辺とを導入する．また，項 e_j の最初のリテラルが否定演算子つきの変数 $\overline{x_i}$ であるときは，まだ使われていない b_{ip} を見つけ，b_{ip} から r_j へ向かう有向辺と u_j から $c_{i,p+1}$ へ向かう有向辺とを導入する．

項 e_j の第 2 のリテラルと第 3 のリテラルについても上と同様の処理を行う．ただし，第 2 のリテラルの場合は，s_j と v_j の 2 頂点を用い，第 3 のリテラルの場合は，t_j と w_j の 2 頂点を用いる．このようして，各々の I_j を，項 e_j に含まれる変数を表す三つの仕掛け H と接続する．接続は，否定演算子のつかないリテラルの場合は c-頂点から来て，一つ下の段の b-頂点へ戻り，否定演算子つきのリテラルの場合は b-頂点から来て，一つ下の段の c-頂点へ戻る．このとき，次が成り立つことを示す．

- 上のように構成されたグラフ G が有向ハミルトン閉路を持つための必要十分条件は，ブール式 E が充足可能であることである．

（十分性）E を充足する真理値割り当て T が存在するとしよう．これに対し，有向ハミルトン閉路を次のように構成する．

(1) H の部分 [図 10.9(b) のグラフ] だけを通過する路を，真理値割り当て T にしたがって構成することから始める．すなわち，閉路は，$T(x_i) = 1$ ならば a_i から b_{i0} へ進み，$T(x_i) = 0$ ならば a_i から c_{i0} へ進む．

(2) 次に，閉路が b_{ip} から $c_{i,p+1}$ へ向かう有向辺を含み，さらに，b_{ip} はまだ閉路の中に取り込まれていない I_j（の一つ）へ向かう有向辺を持つとする．このとき，閉路の中に I_j の 6 個の頂点のすべてを通り，その後，$c_{i,p+1}$ へ戻る「迂回路」を導入する．有向辺 $b_{ip} \to c_{i,p+1}$ は閉路から削除する．ただし，その二つの端点は相変わらず閉路上に残されている．

(3) 同様に，閉路が c_{ip} から $b_{i,p+1}$ へ向かう有向辺を含み，さらに，c_{ip} は I_j へ

向かう有向辺でまだ閉路の中に取り込まれていないものを持つとする．このときも，I_j の 6 個の頂点のすべてを通る「迂回路」を含むように閉路を修正する．

T が E を充足するという仮定により，手順 (1) で最初に構成される路が，手順 (2) や手順 (3) で各項 e_j に対する仕掛け I_j を閉路の中に取り込むことを可能にするための有向辺を少なくとも一つ含むことが保証される．したがって，すべての I_j が閉路の中に含まれることになり，その閉路は有向ハミルトン閉路になる．

（必要性）次に，グラフ G が有向ハミルトン閉路を持つとする．このとき，E は充足可能であることを示す．はじめに，我々がこれまで検討してきた解析から，二つの重要な点を思い出しておく．

(1) ハミルトン閉路が，ある I_j に r_j, s_j または t_j から入るとき，その閉路は，各々，u_j, v_j または w_j から出て行かねばならない．

(2) ハミルトン閉路を，図 10.9(b) のように，仕掛け H を環状にたどって行く路として見るとき，ある I_j に行く寄り道は，有向辺 $b_{ip} \rightarrow c_{i,p+1}$ または $c_{ip} \rightarrow b_{i,p+1}$ の一つに「平行」なもう一つの有向辺のようにみなすことができる．

I_j に行く寄り道を無視して考えるとき，ハミルトン閉路は，H_i だけを使って作られる 2^n 通りの閉路のうちの一つになっている．各 i に対し，a_i から b_{i0} へ進むか，あるいは，c_{i0} へ進むかという選択の可能性があるからである．これらの n 個の選択の組の各々が，E の中に出現する変数に対する真理値割り当てに対応する．もしこれらの選択の組の一つが，I_j まで含めて，ハミルトン閉路を生成するならば，この真理値割り当ては，必ず，E を充足する．

その理由を述べる．閉路が a_i から b_{i0} へ進む場合，I_j への寄り道が可能であるのは j 番目の項が三つのリテラルの一つとして x_i を含むときである．また，閉路が a_i から c_{i0} へ進む場合，I_j への寄り道が可能であるのは j 番目の項が三つのリテラルの一つとして $\overline{x_i}$ を含むときである．したがって，すべての仕掛け I_j が閉路に含まれるという事実から，変数への真理値割り当ては各項の三つのリテラルのうち少なくとも一つを真にすること，すなわち，E は充足可能であることが導かれる．▨

例 10.22　3-CNF 式 $E = (x_1 + x_2 + x_3)(\overline{x_1} + \overline{x_2} + x_3)$ を対象として，非常に簡単な例ではあるが，定理 10.21 の構成の様子を見てみよう．完成したグラフを図 10.10 に示す．H-型の仕掛けと I-型の仕掛けをつなぐ有向辺は点線で示す．点線を用いるのは図を見やすくすることだけが目的であり，それ以外に点線と実線の間には何の違いもない．

たとえば，左上にあるのは，x_1 に対する仕掛けである．x_1 は E の中にその

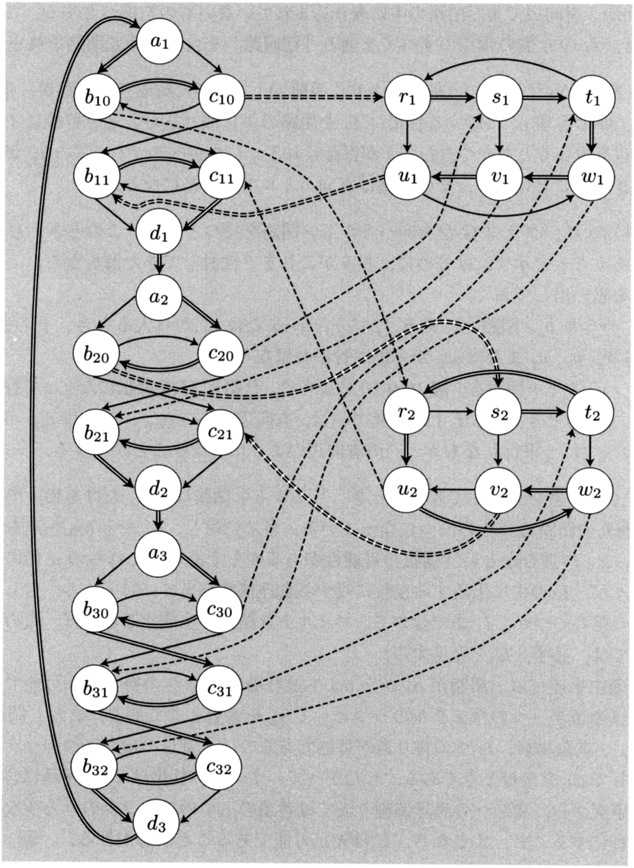


図 10.10　ハミルトン閉路の構成の例

ままの形で 1 回，否定演算子つきで 1 回出現するので，「梯子」は 1 段だけでよい．したがって，b と c は 2 行ある．左下は，x_3 に対する仕掛けである．x_3 は E の中にそのままの形で 2 回出現し，否定演算子つきでは 1 回も出現しない．したがって，これらの項の中での x_3 の使用に対応して I_1 と I_2 に対する仕掛け

を取り付けることを可能にするため，有向辺 $c_{3p} \to b_{3,p+1}$ は 2 通り必要となる．これが，x_3 に対する仕掛けで三つの b-c 行が必要となる理由である．

項 $(\overline{x_1} + \overline{x_2} + x_3)$ に対応する仕掛け I_2 を考えてみよう．第 1 のリテラル $\overline{x_1}$ に対し，b_{10} を r_2 に取り付け，また，u_2 を c_{11} に取り付ける．第 2 のリテラル $\overline{x_2}$ に対しても，b_{20} と s_2，および，v_2 と c_{21} に対して同様のことを行う．第 3 のリテラル x_3 は否定演算子が付いていないので，c の頂点と一つ下段の b の頂点が使われる．すなわち，c_{31} を t_2 に取り付け，w_2 を b_{32} に取り付ける．

E を充足する真理値割り当てはいくつかあるが，その一つは，$x_1 = 1, x_2 = 0, x_3 = 0$ という割り当てである．この割り当てのもとで，第 1 の項は第 1 のリテラル x_1 によって充足され，第 2 の項は第 2 のリテラル $\overline{x_2}$ によって充足される．この真理値割り当てに対し，有向辺 $a_1 \to b_{10}$，$a_2 \to c_{20}$，$a_3 \to c_{30}$ を含むハミルトン閉路を構成することができる．この閉路は，H_1 から I_1 へ寄り道をする，すなわち，有向辺 $c_{10} \to r_1$，を使い，I_1 の中のすべての頂点を巡回し，最後に b_{11} に戻ることにより第 1 の項を処理する．次に，有向辺 $b_{20} \to s_2$ から出発し，I_2 の中のすべての頂点を巡回し，最後に c_{21} に戻るような H_2 から I_2 への寄り道によって第 2 の項が処理される．図 10.10 では，ハミルトン閉路の全体を 2 重線（実線と点線）と大きな矢印とで表示する．　▨

10.4.5　無向ハミルトン閉路問題と TSP

無向ハミルトン閉路問題と巡回セールスマン問題もやはり *NP* 完全であるが，それらの証明は比較的やさしい．我々は，すでに 10.1.4 項で TSP が $\mathcal{NP}$ に属すことを見た．HC は TSP の特別の場合であり，したがって，HC もまた $\mathcal{NP}$ に属す．以下，DHC から HC への還元と，HC から TSP への還元を示す．

問題：無向ハミルトン閉路問題（HC）
入力：無向グラフ G.
出力：G がハミルトン閉路を持つとき，またそのときに限り，“yes” を出力．
還元のもととなる問題：DHC

定理 10.23　無向ハミルトン閉路問題は NP 完全である．

証明　DHC から HC への還元を次のように定める．有向グラフ G_d が与えられたとする．我々が構成する無向グラフを G_u と呼ぶことにしよう．G_d のすべての頂点 v に対し，G_u の中に三つの頂点 v_1, v_2, v_3 を用意する．G_u の辺は次のように定める．

(1)　G_d のすべての頂点 v に対し，G_u の中に辺 $(v^{(0)}, v^{(1)})$, $(v^{(1)}, v^{(2)})$ を定める．

(2)　G_d の中に有向辺 $v \to w$ があるとき，G_u の中に辺 $(v^{(2)}, w^{(0)})$ を定める．

図 10.11 に，有向辺 $v \to w$ に対応する辺を含めた，G_u の辺の様子を示す．

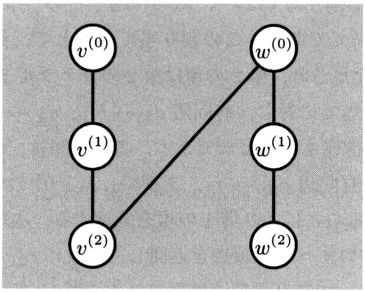


図 10.11　G_d に含まれる有向辺 $v \to w$ は，G_u の中の v の添字が 2 の頂点と w の添字が 0 の頂点を結ぶ辺によって置き換えられる．

明らかに，G_d から G_u を構成するのに要する時間は多項式時間である．次のことを示す．

- G_u がハミルトン閉路を持つための必要十分条件は，G_d が有向ハミルトン閉路を持つことである．

(十分性) $v_1, v_2, \ldots, v_n, v_1$ が有向ハミルトン閉路であるとする．このとき，明らかに，

$$v_1^{(0)}, v_1^{(1)}, v_1^{(2)}, v_2^{(0)}, v_2^{(1)}, v_2^{(2)}, v_3^{(0)}, \ldots, v_n^{(0)}, v_n^{(1)}, v_n^{(2)}, v_1^{(0)}$$

は G_u における無向ハミルトン閉路である．各列を下方に向けて降り，次に，G_d の有向辺にしたがい，次の列の最上位の頂点に飛び移る．この動作を繰り返せばよい．

(必要性) まず，G_u の各頂点 $v^{(1)}$ は二つの辺しか持たないので，ハミルトン閉路の中に現れる際には $v^{(0)}$ と $v^{(2)}$ のうちの一方を直前の頂点とし，もう一方を直後の頂点として，これらと一緒に現れるしかないことに注意する．したがって，G_u の中のハミルトン閉路から各頂点の上付きの添字を抜き出して並べると，必ず，$0, 1, 2, 0, 1, 2, \ldots$

というパターンか，または，その逆の $2, 1, 0, 2, 1, 0, \ldots$ というパターンになる．これらの二つのパターンは，一つの同じ閉路を互いに相異なる方向からたどることに対応するので，パターンは $0, 1, 2, 0, 1, 2, \ldots$ であると仮定しても差し支えない．したがって，閉路に含まれる辺のうち，上付きの添字が 2 の頂点から上付きの添字が 0 の頂点への辺を集めてみると，これらの辺は G_d の有向辺に対応し，有向辺の矢印の向きに次々連なるものであることがわかる．こうして，G_u の無向ハミルトン閉路から G_d の有向ハミルトン閉路が得られる． ▨

問題：巡回セールスマン問題（TSP）

入力：辺が整数の重みを持つ無向グラフ G と限界値 k.

出力：閉路に含まれる辺の重みの和が k 以下であるようなハミルトン閉路を G が持つとき，またそのときに限り，“yes” を出力．

還元のもととなる問題：HC

定理 10.24　巡回セールスマン問題は *NP* 完全である．

証明　HC から TSP への還元を次のように定めればよい．グラフ G が与えられたとき，頂点と辺は G と同じで，各辺が 1 の重みを持つ重みつきグラフを G' とし，G の頂点数 n と同じ値を限界値 k とする．このとき，G' の中に重み n のハミルトン閉路が存在するための必要十分条件は，G の中にハミルトン閉路が存在することである．
▨

10.4.6　*NP* 完全問題のまとめ

図 10.12 に，本章で取り上げたすべての還元の状況を示す．図の中で TSP のような個々の問題からの SAT への還元もすべて指示してあることに注目されたい．これは，定理 10.9 で，すべての多項式時間非決定性チューリング機械によって受理される言語から SAT への還元を示したことによる．はっきりと明示はしなかったものの，これらの TM の中には，TSP を解く TM，IS を解く TM，等々が当然含まれる．したがって，すべての *NP* 完全問題は互いに多項式時間還元可能である．これらのすべての問題は，計算量の観点から見ると，同じ問題を切り口を変えて異なる側面から見たものであるということもできよう．

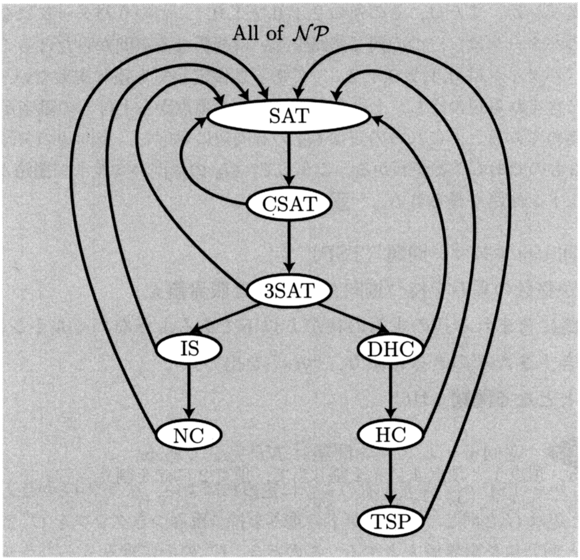


図 10.12　$\mathcal{NP}$ 完全問題の間の変換の様子

10.4.7　10.4 節の演習問題

♣ * **問 10.4.1**　グラフ G の中の **k-クリーク** (k-clique) とは，G に属す k 個の頂点の集合で，すべての（相異なる）2 頂点の間に辺が存在するもののことである．したがって，2-クリークとは，単に，辺で結ばれた 1 対の頂点のことであり，3-クリークは三角形である．次の問題を**クリーク問題** (CLIQUE) という．グラフ G と定数 k が与えられたとき，G が k-クリークを持つかどうかを判定せよ．

a)　図 10.1 のグラフ G に対し CLIQUE の解が "yes" となるような k の最大値を求めよ．

b)　k-クリークに含まれる辺の個数を k の関数として答えよ．

c)　頂点被覆問題を CLIQUE へ還元することにより，CLIQUE は $\mathcal{NP}$ 完全であることを証明せよ．

♣*! **問 10.4.2**　次の問題を**彩色問題** (coloring problem) という．グラフ G と定数 k が与えられたとき，G が "k-彩色可能" であるかどうか，すなわち，どの辺もその 2 端点が同じ色にならないように G の各頂点に k 色の色の一つを割り当てることができるかどうか，を判定せよ．たとえば，図 10.1 のグラフは 3-彩色可能

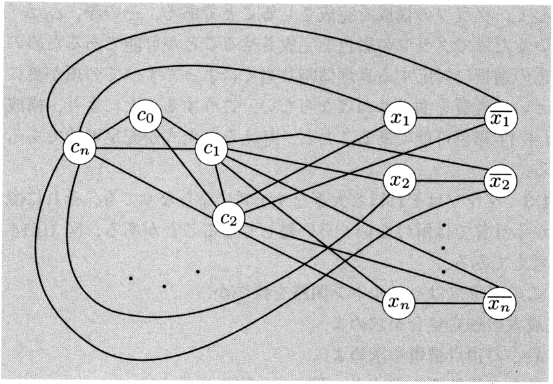


図 10.13　彩色問題が NP 完全であることを示す構成の一部

である．頂点 1 と頂点 4 に赤を割り当て，頂点 2 に緑を割り当て，頂点 3 に青を割り当てればよいからである．一般に，グラフが k-クリークを持つとき，そのグラフは，k 色よりもっと多い種類の色を必要とすることはあり得るが，k より小さい値 k' に対して，k'-彩色可能となることはない．

この演習問題では，彩色問題が NP 完全であることを示すための構成の一部を与える．読者は，残された部分を補い，証明を完成させて頂きたい．還元は 3SAT からの還元である．n 変数の 3-CNF 式が与えられたとする．この式を変換して，その一部が図 10.13 に示されているようなグラフを構成する．図の左側に示されているように，$n+1$ 個の頂点 $c_0, c_1, \ldots, c_n$ を用意し，$(n+1)$-クリークを構成する．したがって，これらの頂点はすべて互いに異なる色で塗られることが必要である．頂点 c_j に割り当てられる色を「c_j の色」と呼ぶことにする．

また，各変数 x_i に対し 2 個の頂点を用意し，それぞれを x_i，$\overline{x_i}$ とみなすことにする．これらの 2 頂点は辺で結ばれる．したがって，これらに同じ色が割り当てられることはない．さらに，x_i に対するこれらの 2 頂点は，いずれも，0 と i 以外のすべての j に対する c_j と結ばれる．この結果，x_i と $\overline{x_i}$ のうちの一方には c_0 の色が塗られ，他方には c_i の色が塗られることになる．c_0 の色が塗られた頂点には「真」の値が割り当てられ，c_i の色が塗られた頂点には「偽」の値が割り当てられたと考える．このようにして，彩色のやり方が一つ選ばれると，それは一つの真理値割り当てに対応する．

読者に求められているのは，ブール式に含まれる各項に対応する部分のグラフ

を設計して，グラフの構成を完成させることである．その際，c_0 から c_n までの色を用いるだけでグラフの彩色を完成させることが可能であるための必要十分条件は，色の選択に対応する真理値割り当てによってすべての項が真になることである，という性質を満たさねばならない．こうすることにより，構成されたグラフが $(n+1)$-彩色可能であることと，与えられた式が充足可能であることとが同値になる．

♣ **! 問 10.4.3** グラフはそれほど大きなサイズにならなくても，それに関する *NP* 完全問題が手計算では解けないくらい難しくなることがある．図 10.14 のグラフについて考えてみる．

*a) このグラフはハミルトン閉路を持つか．

b) 最大の独立集合を求めよ．

c) 最小の頂点被覆を求めよ．

d) 最小の辺被覆を求めよ．(問 10.4.4(b) を見よ．)

e) このグラフは 2-彩色可能であるか．

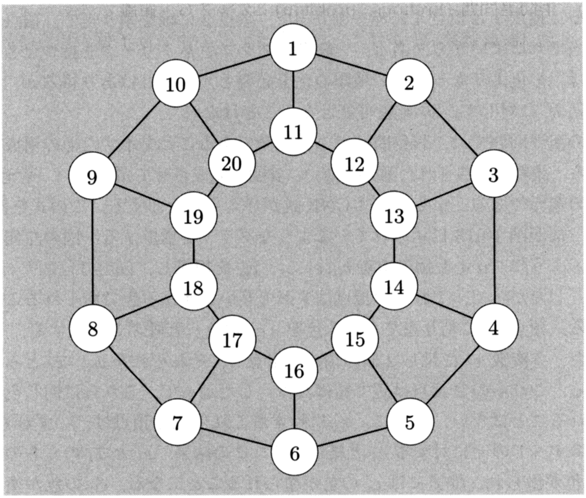


図 10.14 グラフ

♣ **問 10.4.4** 次の問題の各々について，*NP* 完全であることを証明せよ．

a) **部分グラフ同型問題** (subgraph-isomorphism problem)：グラフ G_1, G_2

が与えられたとき，G_1 は G_2 の複製（コピー）を部分グラフとして含むかどうかを判定せよ．

!b) **辺被覆問題** (edge-cover problem)：グラフ G と整数 k が与えられたとき，G は k 個の辺から成る「辺被覆」を持つかどうかを判定せよ．ただし，辺被覆とは，G の辺の集合で，G の各頂点がその集合に属する少なくとも一つの辺の端点となるようなもののことである．

!c) **整数線型計画法** (linear integer programming problem)：整数の定数 a_i, c と変数 $x_1, x_2, \ldots, x_n$ に対し，$\sum_{i=1}^{n} a_i x_i \le c$ または $\sum_{i=1}^{n} a_i x_i \ge c$ の形の線形の制約不等式の集合が与えられたとき，すべての制約不等式を成り立たせるような各変数への整数の割り当てが存在するかどうかを判定せよ．

!d) **支配集合問題** (dominating-set problem)：グラフ G と整数 k が与えられたとき，G の k 個の頂点から成る集合 S で，G のすべての頂点が，S に属すか，または，S に属す頂点に隣接するかのいずれかであるようなものが存在するかどうかを判定せよ．

e) **消防署問題** (firehouse problem)：グラフ G，距離 d,「消防署」の建設予算（設置個数）f が与えられたとき，G の中から f 個の頂点（消防署）を選んで，G のどの頂点もどれかの消防署からの距離が k 以下となるようにすることが可能かどうかを判定せよ．

*!f) $\frac{1}{2}$**-クリーク問題** (half-clique problem)：偶数個の頂点を持つグラフ G が与えられたとき，G の頂点数のちょうど半分の個数の頂点から成る G のクリークが存在するかどうかを判定せよ．(クリークについては問 10.4.1 を見よ.) **ヒント**：CLIQUE を $\frac{1}{2}$-クリーク問題に還元せよ．最大クリークの大きさを調節するために，どのように頂点を追加すればよいかを考えていただきたい．

!!g) **単位実行時間スケジューリング問題** (unit-execution-time-scheduling problem)：k 個の「タスク」

$$T_1, T_2, \ldots, T_k,$$

「プロセッサ」の個数 p，制限時間 t，および，タスクの対の間に定められる $T_i < T_j$ の形のいくつかの「優先度（に関する）制約」が与えられたとき，タスクの実行順序の**スケジュール**で次を満たすものが存在するかどうかを判定せよ．

a) 各々のタスクは，1 と t の間の 1 単位時間に割り当てられる．

b) 1 単位時間に割り当てることのできるタスクの個数は p 以下である．

c) すべての優先度制約を満たす必要がある．すなわち，$T_i < T_j$ とい

う優先度制約があるとき，T_i は T_j より早い単位時間に割り当てられねばならない．

!!h) **直和被覆問題** (exact-cover problem)：集合 S と，S の部分集合の集合 $\{S_1, S_2, \ldots, S_n\}$ が与えられたとき，S の任意の要素 x が T のちょうど一つの要素 S_i に属すような $T \subseteq \{S_1, S_2, \ldots, S_n\}$ が存在するかどうかを判定せよ．

!!i) **ナップザック問題** (knapsack problem)：k 個の（正の）整数 $i_1, i_2, \ldots, i_k$ が与えられたとき，それらを二つのグループに分け，それぞれの和が同じになるようにすることができるかどうかを判定せよ．**注意**：この問題は，上辺だけ見ると $\mathcal{P}$ に属すように思われるかもしれない．その原因は，恐らく，与えられた整数が小さな整数であると仮定してしまうところにある．実際，個々の整数の値が，与えられた整数の個数 k に関するある多項式以下に制限されているような場合には，この問題を解く多項式時間のアルゴリズムが存在する．しかし，各々が 2 進数で表現され，全体の長さが n であるような k 個の整数の中には，値が n に関してほとんど指数であるくらい大きな整数が含まれる場合もあり得るのである．

♣ **問 10.4.5** グラフ G における**ハミルトン路** (Hamilton path) とは，G に属すすべての頂点を，任意の $i = 1, 2, \ldots, k-1$ に対し，v_{n_i} と $v_{n_{i+1}}$ の間に辺が存在するように並べた列 $v_{n_1}, v_{n_2}, \ldots, v_{n_k}$ である．**有向ハミルトン路** (directed Hamilton path) とは，有向グラフに対して同様に定義されたものである．すなわち，各 v_{n_i} から $v_{n_{i+1}}$ への有向辺が存在するような列 $v_{n_1}, v_{n_2}, \ldots, v_{n_k}$ である．ハミルトン路に対して課される条件は，ハミルトン閉路に対して課される条件よりほんのわずかに弱いだけであることに注意されたい．追加の条件として，v_{n_k} と v_{n_1} を結ぶ辺や v_{n_k} から v_{n_1} への有向辺が存在するという条件を付け加えれば，まさに，ハミルトン閉路の条件が得られる．(有向) ハミルトン路問題とは次の問題である．(有向) グラフが与えられたとき，(有向) ハミルトン路が少なくとも一つ存在するかどうかを判定せよ．

***a)** 有向ハミルトン路問題は $\mathcal{NP}$ 完全であることを証明せよ．**ヒント**：DHC からの還元を考えよ．任意の一つの頂点を選び，それを二つの頂点に分割する．これらの 2 頂点は有向ハミルトン路の 2 端点となるべきものであり，また，そのような路が存在するのは，もとのグラフが有向ハミルトン閉路を持つときであり，またそのときに限る，という条件を満たすようにする．

b) (無向) ハミルトン路問題は $\mathcal{NP}$ 完全であることを証明せよ．**ヒント**：定理 10.23 の構成を手直しすることを考えよ．

*!c)　次の問題は NP 完全であることを証明せよ．グラフ G と整数 k が与えられたとき，G が高々k 個の葉頂点から成る極大木を持つかどうかを判定せよ．ヒント：ハミルトン路問題からの還元を考えよ．

!d)　次の問題は NP 完全であることを証明せよ．グラフ G と整数 d が与えられたとき，G が d より大きい次数の頂点を含まない極大木を持つかどうかを判定せよ．(極大木における頂点 v の**次数** (degree) とは，木において v を端点とする辺の個数のことである．)

10.5　第10章の要約

◆**クラス $\mathcal{P}$ と クラス $\mathcal{NP}$**：$\mathcal{P}$ は，入力列の長さの関数として，ある多項式の量の時間で計算を行うテューリング機械によって受理されるすべての言語，または問題，から成るクラスである．$\mathcal{NP}$ は，非決定性の選択の任意の系列に沿う時間に関して多項式の限界を持つ非決定性テューリング機械によって受理されるすべての言語，または問題，から成るクラスである．

◆**$\mathcal{P}=\mathcal{NP}$ 問題**：$\mathcal{P}$ と $\mathcal{NP}$ が実は同じクラスであるかどうかは，まだわかっていない．しかし，$\mathcal{NP}$ には属すが $\mathcal{P}$ には属さない言語が存在することを，我々は強く予想する．

◆**多項式時間還元**：ある問題の入力例を，別の問題で同じ答 — yes または no — を持つものに多項式時間で還元できるとき，第一の問題は第二の問題に多項式時間還元可能であるという．

◆***NP* 完全問題**：言語が NP 完全であるとは，その言語が $\mathcal{NP}$ に属し，かつ，$\mathcal{NP}$ に属す任意の言語からその言語への多項式時間還元が存在することである．我々は，NP 完全問題はどれも $\mathcal{P}$ に属さないことを強く信じている．NP 完全であることが知られている数千個にも及ぶ問題のいずれについても，まだだれも多項式時間のアルゴリズムを発見していないという事実は，NP 完全問題はどれも $\mathcal{P}$ に属さないという予想を相互に強め合う「証拠」となる．

◆***NP* 完全な充足可能性問題**：クックの定理により，最初の NP 完全問題 — ブール式が充足可能であるかどうかを問う問題 — を提示された．証明の方針は，$\mathcal{NP}$ に属すすべての問題は多項式時間で SAT 問題に還元可能であることを示すことであった．さらに，入力のブール式を，各々が 3 個のリテラルだけから成るいくつかの項の積に限定した場合 — 3SAT 問題 — でも，この問題の NP 完全性は保たれる．

◆**その他の *NP* 完全問題**：現在までに，広大な分量の NP 完全問題が知られている．それらは，それ以前に NP 完全であることが証明された問題からその問題への多項式時間還元を構成することによって，NP 完全であることが証明される．我々は，適当な還元を与えることにより，以下の問題が NP 完全であることを示した．独立集

合問題，頂点被覆問題，有向グラフおよび無向グラフに対するハミルトン閉路問題，巡回セールスマン問題.

10.6 第10章の参考文献

NP 完全性の概念を，ある問題が多項式時間では解くことができないことを示す証拠として導入したのは，SAT，CST，3SAT が *NP* 完全であることを示す証明と同様，S. C. Cook [3] である．それに引き続く論文 R. M. Karp [6] も，一般にそれと同等の重要さを持つものとして認識されている．*NP* 完全性は，単に孤立した現象ではなく，オペレーションズ・リサーチをはじめとする種々の分野の研究者たちが数十年にわたり研究してきた難しい組合せ問題のうちの非常に多くの問題に共通に適用される性質であることを示した論文だからである．10.4 節で *NP* 完全であることが証明された問題，すなわち，独立集合問題，頂点被覆問題，ハミルトン閉路問題，そして，巡回セールスマン問題は，その論文から採ったものである．そればかりでなく，演習問題として与えた問題のうち，クリーク問題，辺被覆問題，ナップザック問題，彩色問題，そして，直和被覆問題などに対する答も，その論文の中に見出すことができる．

M. R. Garay と P. S. Johnson [4] の書物は，どの問題が *NP* 完全であるか，また，多項式時間で解ける特別の場合はどんな場合か，というようなことに関して，その当時知られていた事柄の集大成である．[5] には，*NP* 完全問題に対する近似解を多項式時間で求めることに関する論文が収められている．

NP 完全性の理論に対して貢献をした論文として，上記の他に，次のようなものが挙げられる．テューリング機械の計算時間によって定義される言語のクラスの研究は，J. Hartmanis と R. E. Stearns [8] を嚆矢とする．A. Cobham [2] は，それまでの，たとえば $O(n^2)$ のような個別の多項式時間のアルゴリズムという考え方を一歩進めて，クラス $\mathcal{P}$ の概念を抽出し明確にした最初の論文である．L. A. Levin [7] は，時期は少し遅れたものの，独立に *NP* 完全性のアイデアを発見した．

整数線型計画法 [問 10.4.4(c)] の *NP* 完全性は [1] に示されているが， J. Gathen と M. Sieveking の未刊行のノートにも収録されている．単位実行時間スケジューリング問題 [問 10.4.4(g)] の *NP* 完全性は [9] による．

[1] I. Borosh and L. B. Treybig, “Bounds on positive integral solutions of linear Diophantine equations,” *Proceedings of the AMS* **55** (1976), pp.299–304.

[2] A. Cobham, “The intrinsic computational difficulty of functions,” *Proc. 1964 Congress for Logic, Mathematics, and the Philosophy of Science*, North Holland, Amsterdam, pp.24–30.

[3] S. C. Cook, “The complexity of theorem-proving procedures,” *Third ACM*

Symposium on Theory of Computing (1971), ACM, New York, pp.151–158.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, H. Freeman, New York, 1979.

[5] D. S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Co., 1996.

[6] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations* (R. E. Miller, ed.), Plenum Press, New York, pp.85–104.

[7] L. A. Levin, "Universal sorting problems," *Problemi Peredachi Informatsii* **9**:3 (1973), pp.265–266.

[8] J. Hartmanis and R. E. Stearns, "On the computational complexity of algorithms," *Transactions of the AMS* **117** (1965), pp.285–306.

[9] J. D. Ullman, "NP-complete scheduling problems," *J. Computer and System Science* **10**:3 (1975), pp.384–393.

11 その他の「問題のクラス」

実行不能な問題の物語は，$\mathcal{NP}$ に始まり，$\mathcal{NP}$ に終わる，という訳ではない．実行不能と思われる「問題のクラス」や，別の何らかの理由で興味が持たれている問題のクラスは，他にもたくさんある．それらのクラスにまつわる疑問点のいくつかは，$\mathcal{P} = \mathcal{NP}$ 問題と同様に，まだ未解決のまま残されている．

我々はまず，$\mathcal{P}$ と $\mathcal{NP}$ に密接な関係を持つ一つのクラスから始めることにしよう．それは，$\mathcal{NP}$ に属す言語の補集合全体から成るクラスである．このクラスは，通常，"補 $\mathcal{NP}$" と呼ばれる．もし，$\mathcal{P} = \mathcal{NP}$ が成り立つとすれば，補 $\mathcal{NP}$ もそのクラスに一致する．$\mathcal{P}$ は補集合をとる演算に関して閉じているからである．しかし，実際は，そのようなことは起こり得ず，補 $\mathcal{NP}$ はこれらの二つのクラスのいずれとも異なるように思われる．また，どの NP 完全問題も補 $\mathcal{NP}$ には属さないように思われる．

次に，我々は，$\mathcal{PS}$ と呼ばれるクラスについて考察する．これは，入力の長さに関して多項式で表される程度のテープ量しか使わないテューリング機械によって解くことのできる問題の全体から成るクラスである．これらのテューリング機械は，ヘッドがテープ上のある限られた領域にとどまる限り，指数関数で表されるような量の時間を使うことが許される．多項式時間の場合とは対照的に，制約が多項式領域である場合には，非決定性のモデルを考えても TM の能力は増加しないことを証明することができる．また一方，$\mathcal{PS}$ が $\mathcal{NP}$ に属す問題のすべてを含むことは明らかであるが，$\mathcal{PS}$ が $\mathcal{NP}$ に一致するかどうか，さらには，$\mathcal{PS}$ が $\mathcal{P}$ に一致するかどうかさえ，まだわかっていない．ただし，我々は，$\mathcal{PS}$ はそれらのどちらとも一致しないと予想している．$\mathcal{PS}$ に対して完全であり，かつ，$\mathcal{NP}$ には属さないように見える問題の例を与える．

その次に，我々は，ランダム・アルゴリズムに目を転じ，$\mathcal{P}$ と $\mathcal{NP}$ との中間に位置する二つの言語のクラスを取り上げる．第 1 のクラスは，$\mathcal{RS}$ と呼ばれる「ランダム多項式」の言語のクラスである．このクラスに属す言語は，「コ

イン投げ」または（実際には）乱数生成ルーチンを用いることが許されていて，多項式時間で実行されるアルゴリズムを持つ．アルゴリズムは，与えられた入力の，対象とする言語への所属を確認するか，または，「わからない」という答を出力する．さらに，入力が言語に属す場合，アルゴリズムは，0 より大きいある確率で正しい答を与える．したがって，このアルゴリズムを繰り返し適用することにより，所属しないことが確認される確率は 1 に近づいていく．

第 2 のクラスは，$\mathcal{ZPP}$ (zero-error, probabilistic polynomial) と呼ばれ，やはり，乱数に関わるクラスである．ただし，このクラスに属す言語に対するアルゴリズムは，入力がその言語に属すかどうかについて，必ず “yes” または “no” を答える．アルゴリズムの平均実行時間は多項式時間である．しかし，どんな多項式の限界によって許容されるよりも多くの時間を要する場合があり得る．

これらの概念を結びつけるため，素数性判定という大事な問題を考える．今日の多くの暗号系は，次の 2 点に依存している．

(1) 大きな素数を速やかに発見する能力（これは，外部から盗聴されないようにして機械の間の通信を許すためである）

(2) 整数を 2 進数として表したときの長さ n の関数として計算時間を測るとき，整数の因数分解には指数時間を要するという仮定

我々は，素数性判定問題は $\mathcal{NP}$ にも，また，補 $\mathcal{NP}$ にも属すことを見ていく．このことから，素数性判定問題について NP 完全であることを証明できるとはとても思えないことがわかる．NP 完全性の証明は，ある問題がほとんど確実に指数時間を必要とするということを主張するために，もっとも普通に用いられる論法であることを考えると，これは残念なことである．我々はまた，素数性判定問題はクラス $\mathcal{RP}$ に属すことも見ていく．この事実には，良い点もあり，また，悪い点もある．良い点は，実用的に，暗号系が素数を必要とするとき，実際に $\mathcal{RP}$ に属すアルゴリズムを用いてそれらを求めることができることである．悪い点は，素数性判定問題の NP 完全性を証明することはできないであろうという仮定を，さらに強化するものになることである．

11.1 $\mathcal{NP}$ に属す言語の補集合

言語のクラス $\mathcal{P}$ は，補集合をとる演算のもとで閉じている．（演習問題 10.1.6 を見よ．）理由を簡単に述べてみよう．L を $\mathcal{P}$ に属す言語とし，M を L を受理する TM とする．$\overline{L}$ を受理するため，M を次のように変更する．新しい受理状態 q を導入し，M の受理状態ではない状態から q への新しい状態遷移を定め

る．その上で，M の本来の受理状態をすべて非受理状態に変える．このようにすると，修正された TM は $\overline{L}$ を受理する．計算時間は M と同じであるか，または，それより 1 ステップ多い時間である．したがって，L が $\mathcal{P}$ に属すならば，$\overline{L}$ も $\mathcal{P}$ に属す．

$\mathcal{NP}$ が補集合をとる演算のもとで閉じているかどうかは，まだわかっていない．しかし，$\mathcal{NP}$ は補集合をとる演算のもとで閉じていないように思われる．とくに，言語が NP 完全であるとき，その補集合は $\mathcal{NP}$ に属さないと予想されている．

11.1.1　言語のクラス 補 $\mathcal{NP}$

補 $\mathcal{NP}$ とは，補集合が $\mathcal{NP}$ に属すような言語全体の集合である．11.1 節の冒頭で，$\mathcal{P}$ に属すすべての言語に対し，その補集合もまた $\mathcal{P}$ に属すことを見た．したがって，それらは当然 $\mathcal{NP}$ に属す．これに対し，我々は，どの NP 完全問題に対しても，その補集合が $\mathcal{NP}$ に属すことはなく，したがって，どの NP 完全問題も 補 $\mathcal{NP}$ には属さないことを信じる．同様に，NP 完全問題の補集合について，それらは定義により補 $\mathcal{NP}$ に属すが，$\mathcal{NP}$ には属さないことを信じる．図 11.1 に，$\mathcal{P}$，$\mathcal{NP}$，補 $\mathcal{NP}$ の三つのクラスの間の関係で，我々が信ずる状況を示す．ただし，仮に $\mathcal{P}$ と $\mathcal{NP}$ が一致するとすれば，これらの三つのクラスはすべて同じクラスになってしまうことを記憶に留めておいていただきたい．

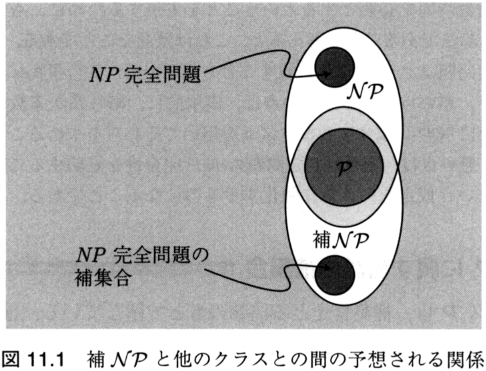


図 11.1　補 $\mathcal{NP}$ と他のクラスとの間の予想される関係

例 11.1 言語 SAT の補集合を考えて見よう．これは，確かに，補 $\mathcal{NP}$ の要素である．この問題を **USAT** (unsatisfiable) と呼ぶことにする．USAT に属す文字列の中には，充足可能でないブール式を符号化した文字列がすべて含まれる．しかし，それだけではなく，USAT の中には，正しいブール式の符号化に対応しない文字列もすべて含まれる．なぜならば，それらはどれも SAT に属さないことが明らかだからである．我々は，USAT は $\mathcal{NP}$ に属さないと信じるが，しかし，その証明はまだ得られていない．

我々が，補 $\mathcal{NP}$ には属すが，$\mathcal{NP}$ には属さないと予想するもう一つの問題の例は，TAUT である．TAUT とは，**恒真式** (tautology)，すなわち，すべての真理値割り当てに対して真であるようなブール式，を符号化した文字列の全体から成る集合である．ブール式 E が恒真式であるための必要十分条件は，$\neg E$ が充足可能ではないことである．したがって，TAUT と USAT は，ブール式 E が TAUT に属すならば，必ず，$\neg E$ は USAT に属し，また，その逆も成り立つ，という関係にある．ただし，USAT には，正しいブール式に対応しない文字列も含まれるが，一方，TAUT に含まれる文字列はすべて正しいブール式である．▨

11.1.2 NP 完全問題と補 $\mathcal{NP}$

$\mathcal{P} \neq \mathcal{NP}$ と仮定しよう．その場合でも，補 $\mathcal{NP}$ をめぐる状況は，必ずしも，図 11.1 に示されている通りではない可能性が残されている．$\mathcal{NP}$ と補 $\mathcal{NP}$ が相等しく，それらは $\mathcal{P}$ より真に大きい，ということがあり得るからである．すなわち，USAT や TAUT のような問題が，非決定性多項式時間で解くことはできる（すなわち，それらは $\mathcal{NP}$ に属す）が，それらを決定性多項式時間で解くことはできない，ということが判明するかも知れない．しかし，これまで我々は，補集合が $\mathcal{NP}$ に属すような NP 完全問題をただの一つも見つけることができないでいる．この事実は，次の定理で証明するように，$\mathcal{NP} \neq$ 補 $\mathcal{NP}$ が成り立つことを示す強い証拠のように思える．

定理 11.2 $\mathcal{NP} =$ 補 $\mathcal{NP}$ であるための必要十分条件は，補集合が $\mathcal{NP}$ に属すような NP 完全問題が存在することである．

証明 （必要性）$\mathcal{NP}$ と補 $\mathcal{NP}$ が相等しいならば，任意の NP 完全問題 L は，$\mathcal{NP}$

に属すことから，明らかに，補 $\mathcal{NP}$ にも属す．ところで，補 $\mathcal{NP}$ に属す問題の補集合は $\mathcal{NP}$ に属すので，L の補集合は $\mathcal{NP}$ に属す．
（十分性）P を，その補集合 $\overline{P}$ が $\mathcal{NP}$ に属すような NP 完全問題とする．このとき，$\mathcal{NP}$ に属すすべての言語 L に対し，L から P への多項式時間還元が存在する．その同じ還元は，$\overline{L}$ から $\overline{P}$ への多項式時間還元でもある．以下，両方向の包含関係を示すことにより，$\mathcal{NP}$ = 補 $\mathcal{NP}$ を証明する．

$\mathcal{NP} \subseteq$ 補 $\mathcal{NP}$：　L を $\mathcal{NP}$ に属す任意の言語とする．このとき，$\overline{L}$ は補 $\mathcal{NP}$ に属す．$\overline{L}$ から $\overline{P}$ への多項式時間還元と，存在が仮定されている $\overline{P}$ に対する非決定性で多項式時間のアルゴリズムを組み合わせると，$\overline{L}$ に対する非決定性で多項式時間のアルゴリズムが得られる．したがって，$\mathcal{NP}$ に属す任意の L に対し，$\overline{L}$ もまた $\mathcal{NP}$ に属す．よって，L は $\mathcal{NP}$ に属す言語の補集合となり，補 $\mathcal{NP}$ に属す．以上より，$\mathcal{NP} \subseteq$ 補 $\mathcal{NP}$ が証明された．

補 $\mathcal{NP} \subseteq \mathcal{NP}$：　L を補 $\mathcal{NP}$ に属す任意の言語とする．このとき，P は NP 完全であり，また，$\overline{L}$ は $\mathcal{NP}$ に属すので，$\overline{L}$ から P への多項式時間還元が存在する．この還元は，また，L から $\overline{P}$ への多項式時間還元でもある．$\overline{P}$ は NP に属すので，この還元を，$\overline{P}$ に対する非決定性で多項式時間のアルゴリズムと組み合わせることにより，L は $\mathcal{NP}$ に属すことが示される．▨

11.1.3　11.1 節の演習問題

♣ **!問 11.1.1**　いくつかの問題を下に掲げる．それぞれについて，その問題は $\mathcal{NP}$ に属すかどうかを答えよ．また，補 $\mathcal{NP}$ に属すかどうかを答えよ．与えられた問題についてその補集合を記述せよ．与えられた問題，または，その補集合が NP 完全である場合には，そのことも証明せよ．

- *a)　問題 TRUE-SAT：すべての変数の値が真であるとき真になるようなブール式 E が与えられたとき，それ以外の（すなわち，すべての変数の値が真である場合以外の）真理値割り当てで E の値を真にするものがあるかどうかを答えよ．
- b)　問題 FALSE-SAT：すべての変数の値が偽であるとき偽になるようなブール式 E が与えられたとき，それ以外の（すなわち，すべての変数の値が偽である場合以外の）真理値割り当てで E の値を偽にするものがあるかどうかを答えよ．
- c)　問題 DOUBLE-SAT：ブール式 E が与えられたとき，E の値を真にする真理値割り当てが二つ以上あるかどうかを答えよ．
- d)　問題 NEAR-TAUT：ブール式 E が与えられたとき，E の値を偽にする真理値割り当ては高々一つしかないかどうかを答えよ．

♣*! **問 11.1.2**　n ビット整数の集合から n ビット整数の集合への 1 対 1 関数 f で，次の性質を満たすものが存在したと仮定する．

1. $f(x)$ は多項式時間で計算することができる．
2. $f^{-1}(x)$ は多項式時間では計算することができない．

このとき，

$$f^{-1}(x) < y$$

を満たす整数の対 (x, y) から成る言語は，$(\mathcal{NP} \cap$ 補 $\mathcal{NP}) - \mathcal{P}$ に属すことを示せ．

11.2　多項式領域で解ける問題

さて，$\mathcal{NP}$ に属す問題をすべて含み，さらに，まだ証明されてはいないが，それ以外の問題も含むように思える一つの問題のクラスを見ていこう．このクラスは，テューリング機械に，計算時間をどれだけ使うかということには関係なく，入力列の長さに関して多項式程度の領域量の使用を許すことによって定義される．最初は，多項式領域の決定性 TM によって受理される言語のクラスと非決定性 TM によって受理される言語のクラスを区別する．しかし，すぐに我々は，これらの二つのクラスは同じであることを見ることとなる．

多項式領域で解ける問題のクラスに対する完全問題 P が存在する．これは，このクラスに属すすべての問題は P に多項式**時間**で還元可能である，という意味である．したがって，仮に，完全問題 P が $\mathcal{P}$ または $\mathcal{NP}$ に属すとすれば，多項式領域有界の TM で受理されるすべての言語もまた，それぞれ，$\mathcal{P}$ または $\mathcal{NP}$ に属すことになる．そのような完全問題の例を一つ紹介する．それは，「限定記号付きブール式」に関する問題である．

11.2.1　多項式領域有界テューリング機械

多項式領域有界テューリング機械の様子を図 11.2 に示す．長さ n の入力 w が与えられたとき，この TM はテープ上の $p(n)$ 個を超える個数のます目を訪れることはない，という性質を持つ多項式 $p(n)$ が存在する．定理 8.12 により，テープは半無限であること，そして，TM は入力の左端の位置からさらに左方に動くことはないことを仮定してよい．

ある多項式領域有界決定性テューリング機械 M に対し $L(M)$ となる言語の全体から成るクラスを，$\mathcal{PS}$ (**多項式領域**，polynomial space) と定める．ま

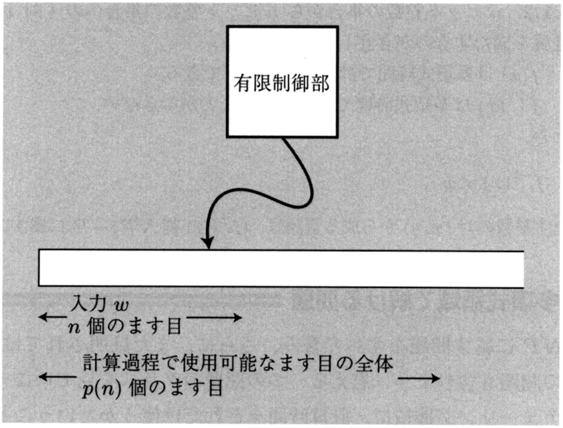


図 11.2　多項式領域を使用する TM

た，ある非決定性の多項式領域有界テューリング機械 M に対し $L(M)$ となる言語の全体から成るクラスを，$\mathcal{NPS}$ (**非決定性多項式領域**，nondeterministic polynomial space) と定める．明らかに，$\mathcal{PS} \subseteq \mathcal{NPS}$ が成り立つ．すべての決定性 TM は非決定性 TM とみなせるからである．ところが，それだけでなく，我々は $\mathcal{PS} = \mathcal{NPS}$ という驚くべき結果を証明することができる[1]．

11.2.2　$\mathcal{PS}$ と $\mathcal{NPS}$ の既知のクラスとの関係

まず，すぐにわかることとして，$\mathcal{P} \subseteq \mathcal{PS}$ と $\mathcal{NP} \subseteq \mathcal{NPS}$ という関係がある．この包含関係が成り立つ理由は，TM が多項式の回数の動作しか行わない場合には，その TM は多項式を超える個数のます目を使うことはできない，もう少し正確にいうと，動作回数プラス 1 より多い個数のます目を訪れることはできない，ということによる．さらに，$\mathcal{PS} = \mathcal{NPS}$ が証明されれば，三つのクラスの間には包含関係の鎖 $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PS}$ が形成されることがわかる．

[1] このテーマを扱った別の文献では，このクラスは PSPACE と書かれていることに読者は気づかれるであろう．しかし，我々は，決定性（または，非決定性）多項式領域有界テューリング機械で解くことのできる問題のクラスを表すのに，スクリプト体で書かれた $\mathcal{PS}$ という表記法を用いることにする．等式 $\mathcal{PS} = \mathcal{NPS}$ が証明されたのちは，$\mathcal{NPS}$ という表記の使用はやめる．

多項式領域有界 TM が有する本質的な特性は，同じ ID が重複して出現する前に行うことのできる動作回数は高々指数個でしかない，という点にある．$\mathcal{PS}$ に関する別のいくつかの興味深い事実を証明する際に，また，$\mathcal{PS}$ は帰納的言語，すなわち，アルゴリズムを持つ言語，しか含まないということを示す際に，我々はこの事実を必要とする．$\mathcal{PS}$ や $\mathcal{NPS}$ の定義の中には，TM に停止することを要求するような事柄は何も含まれていないことに注意されたい．TM は，そのテープ上の多項式の大きさの領域から抜け出すことなく，永久に動き続けることが可能である．

定理 11.3　M を多項式領域有界（決定性，または，非決定性）TM とし，$p(n)$ を領域量の上界を示す多項式とする．このとき，M が長さ n の入力 w を受理するときは必ず $c^{1+p(n)}$ 回以下の動作で受理する，という性質を満たす定数 c が存在する．

証明　証明の本質は，M は，$c^{1+p(n)}$ 回を超える動作を行おうとするとその前に必ずある ID を繰り返さなければならなくなる，という点にある．もし M がある ID を繰り返し経由し，その後に入力を受理したとすると，それよりももっと短い ID の列で受理に至るものがなければならない．すなわち，初期 ID α，繰り返し出現する ID β，そして，受理 ID γ に対し，$\alpha \overset{*}{\vdash} \beta \overset{*}{\vdash} \beta \overset{*}{\vdash} \gamma$ であるとすれば，$\alpha \overset{*}{\vdash} \beta \overset{*}{\vdash} \gamma$ はもっと短い ID の列であって，やはり受理に導くものとなる．

上のような c が必ず存在することを示す議論には，TM によって使用される領域が限られているとき，出現し得る ID の個数には限界がある，という事実が用いられる．もう少し詳しくいうと，次のようになる．M のテープ記号の個数を t とし，M の状態数を s とする．このとき，テープのます目が $p(n)$ 個だけしか使用されないとすると，M の相異なる ID の個数は高々 $sp(n)t^{p(n)}$ 個である．状態は s 個のうちの一つが選ばれ，テープヘッドは $p(n)$ 個のテープ位置のいずれかに置かれ，$p(n)$ 個のます目はテープ記号から成る $t^{p(n)}$ 通りの列のどれかで埋められるからである．

$c = s + t$ とする．$(t+s)^{1+p(n)}$ の 2 項展開を考えると次のようになる．

$$t^{1+p(n)} + (1+p(n))st^{p(n)} + \cdots$$

ここで，第 2 項は $sp(n)t^{p(n)}$ 以上であることに注意すると，$c^{1+p(n)}$ は M のとり得る ID の個数以上であることが証明される．M が長さ n の入力 w を受理するならば，どの ID も重複して現れることのない動作の列によって受理する，ということに注意すれば証明が完了する．したがって，M は，相異なる ID の個数，すなわち，$c^{1+p(n)}$ 以下の長さの動作の列によって入力を受理する．　▨

定理 11.3 を用いて，任意の多項式領域有界 TM を，それと同値であって，高々指数回の動作を行った後に必ず停止する TM に変換することができる．変換の本質的な部分は，我々は，TM が指数回以下の動作で受理することを知っているので，何回の動作が実行されたかを数えることが可能であり，したがって，受理せずに充分な回数の動作を行ったことがわかったら，その時点で TM を停止させてしまうことができる，というところにある．

定理 11.4　L を $\mathcal{PS}$（または，$\mathcal{NPS}$）に属す任意の言語とする．このとき，L は，ある多項式 $q(n)$ と定数 $c > 1$ に対し，高々$c^{q(n)}$ 回の動作の後に必ず停止する多項式領域有界な決定性（または，非決定性）TM によって受理される．

証明　ここでは，決定性 TM に対してのみ証明する．NTM に対しても同様の議論を適用することができる．我々は，L は多項式 $p(n)$ の領域量を持つ TM M_1 によって受理されることを知っている．したがって，定理 11.3 により，M_1 が入力 w を受理するならば，M は高々$c^{1+p(|w|)}$ ステップで w を受理する．

2 本のテープを持つ新しい TM M_2 を設計しよう．M_2 は，第 1 テープを使って M_1 の動作を模倣し，一方，第 2 テープを使って，基数を c として，$c^{1+p(|w|)}$ まで一つずつ数えていく．もし M_2 の数え上げがこの値まで達したら，M_2 は入力を受理せずに停止する．したがって，M_2 は，第 2 テープのます目を $1 + p(|w|)$ 個を超えて使用することはない．また，我々は，M_1 がテープ上で $p(|w|)$ 個より多くのます目を使うことはないと仮定したので，M_2 は，第 1 テープについても $p(|w|)$ 個より多くのます目を使うことはない．

M_2 を 1 テープ TM M_3 に変換する場合も，長さ n の任意の入力に対し $1 + p(n)$ 個より多くのます目を使わないですむように M_3 を構成することができる．M_3 の計算時間は M_2 の計算時間の 2 乗くらいになるかも知れないが，それは $O(c^{2p(n)})$ を超えるものではない[2)]．M_2 は，ある定数 d に対し，$dc^{2p(n)}$ 回以下の動作しか行わないので，$q(n) = 2p(n) + \log_c d$ とすればよい．こうすると，M_3 のステップ数は高々$c^{q(n)}$ 回となる．M_2 は任意の入力に対して必ず停止するので，M_3 も必ず停止する．M_1 は L を受理するので，M_2 と M_3 も L を受理する．よって，M_3 は定理の主張を満足する．　▨

---

2) 実は，定理 8.10 に示された一般的方法は，我々が得ることができる最強の結果ではない．なぜならば，どのテープについても高々$1 + p(n)$ 個のます目しか使用しないので，多テープから 1 テープへの構成の際に，模倣される複数個のテープヘッドは高々$1 + p(n)$ 個のます目分しか互いに離れることはない．したがって，多テープ TM M_2 の $c^{1+p(n)}$ 回の動作は $O(p(n)c^{p(n)})$ ステップで模倣される．これは，上記の値 $O(c^{2p(n)})$ より小さい．

11.2.3 決定性多項式領域と非決定性多項式領域

$\mathcal{P}$ と $\mathcal{NP}$ の間の比較が大層難しい問題であることを知っているので，$\mathcal{PS}$ と $\mathcal{NPS}$ の間の比較が容易に解決されてしまうことには驚かされる．それらは，実は，同じクラスであることが示される．証明は，多項式の領域量 $p(n)$ を持つ非決定性 TM を，多項式の領域量 $O(p^2(n))$ を持つ決定性 TM によって模倣することによって行われる．

証明の核心は，NTM N が ID I から ID J まで高々m 回の動作で移ることが可能かどうかを調べる決定性の再帰的テストにある．DTM D は，すべての ID K に対し，I から K まで $m/2$ 回の動作で移ることができ，かつ，K から J まで $m/2$ 回の動作で移ることができるかどうかを，体系的に試す．すなわち，高々 m 回の動作で $I \overset{*}{\vdash} J$ が成り立つかどうかを決定する再帰的関数 $reach(I, J, m)$ があると考えるとよい．

D のテープを，*reach* への再帰呼び出しの引数を置くスタックとみなす．す

```
BOOLEAN FUNCTION reach(I,J,m)
    ID: I,J; INT: m;
    BEGIN
        IF (m == 1) THEN /* 基礎 */ BEGIN
            I==J であるか，または，I から J へ 1 回の動作で移ることが
            できるかどうかを調べる;
            RETURN TRUE yes の場合，FALSE no の場合;
        END;
        ELSE /* 再帰的な部分 */ BEGIN
            FOR 各々の可能な ID K DO
                IF (reach(I,K,m/2) AND reach(K,J,m/2)) THEN
                    RETURN TRUE;
            RETURN FALSE;
        END;
    END;
```

図 11.3 再帰的関数 *reach* は，指定された回数以内の動作で，ある ID から別の ID に移れるかどうかをテストする

なわち，D は一つの**スタック枠** (stack frame) に $[I, J, m]$ を置く．*reach* によって実行されるアルゴリズムの概要を図 11.3 に示す．

ここで肝要なのは，*reach* は自分自身を 2 回呼び出すが，それらの呼び出しは逐次的に行われるので，各時点で記憶しておく必要があるのは 1 回分の呼び出しだけである点に注意することである．すなわち，たとえば，スタック枠 $[I_1, J_1, m]$ から出発するとき，計算途中のどの時点でも，記憶する必要があるのは，ただ一つの $[I_2, J_2, m/2]$ と，ただ一つの $[I_3, J_3, m/4]$ と，ただ一つの $[I_4, J_4, m/8]$ と，という具合で，最後は第 3 引数が 1 になるところまで進むが，それらだけである．第 3 引数が 1 の時点で，*reach* は基礎ステップを適用することができ，それ以上の再帰呼び出しは必要がなくなる．基礎ステップでは，$I = J$ であるか，または，$I \vdash J$ であるかをテストし，そのどちらかが成り立つ場合には `TRUE` を返し，どちらも成り立たない場合は `FALSE` を返す．図 11.4 に，最初の動作回数として m が与えられたとき，*reach* への呼び出しの個数が最大になる場合の DTM D のスタックの様子を示す．

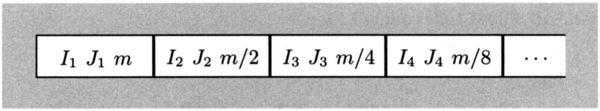


図 11.4　*reach* への再帰呼び出しによって NTM を模倣する DTM のテープの様子

ここで，*reach* への呼び出しの回数は非常に多く必要となり，したがって，図 11.4 のテープの使用量は非常に長くなるように思えるかもしれない．しかし，その長さは「長過ぎて困る」ほど長くはならないことを示す．すなわち，動作回数が m で出発するとき，計算のどの時点においても，テープ上には高々 $\log_2 m$ 個のスタック枠しか現れることがない．定理 11.4 により，NTM N は $c^{p(n)}$ 回を超える回数の動作を行うことはないことが保証されているので，m としてそれより大きな値を考える必要はない．したがって，必要なスタック枠の個数は高々 $\log_2 c^{p(n)}$ であり，この値は $O(p(n))$ である．以上述べたことが，次の定理の証明の背後にある本質的な部分である．

定理 11.5　（サビッチ (Savitch) の定理）$\mathcal{PS} = \mathcal{NPS}$

証明　すべての DTM は NTM の特殊な場合とみなすことができるので，$PS \subseteq NPS$ は明らかである．そこで，$NPS \subseteq PS$ だけを示せばよい．すなわち，L が，ある多項式 $p(n)$ に対し，領域量 $p(n)$ の NTM N によって受理されるならば，L は，ある別の多項式 $q(n)$ に対し，領域量 $q(n)$ の DTM D によっても受理されることを示せばよい．実際，$q(n)$ は，$p(n)$ の 2 乗のオーダの多項式として選ぶことができる．

まず，定理 11.3 により，N が入力を受理する場合，ある定数 c に対し，$c^{1+p(n)}$ ステップ以内に受理すると仮定してよい．長さ n の入力 w が与えられたとき，D は，3 項組 $[I_0, J, m]$ をテープ上に置き，これらの引数について $reach$ を呼び出すことを繰り返すことにより，N が入力 w に対して何を行うかを発見していく．ただし，I_0，J，m は次の通りである．

(1)　I_0 は，入力 w に対する N の初期 ID である．

(2)　J は，テープのます目を高々 $p(n)$ 個使用する任意の受理 ID である．この条件を満たす相異なる J を，D は作業用テープを用いて，体系的に順に生成していく．

(3)　$m = c^{1+p(n)}$.

上で説明したように，同時に記憶しておく必要のある再帰呼び出しの個数は $\log_2 m$ を超えることはない．第 3 引数が m のものが一つ，$m/2$ のものが一つ，$m/4$ のものが一つ，という具合に進み，第 3 引数が 1 のものまでを考えれば十分だからである．したがって，同時にスタック上に置かれるスタック枠の個数が $\log_2 m$ を超えることはない．そして，$\log_2 m$ は $O(p(n))$ である．

さらに，スタック枠自体の領域量は $O(p(n))$ である．その理由を述べる．二つの ID は，書き下すためにそれぞれ $1 + p(n)$ 個のます目しか必要としない．また，m は 2 進数で表すと $\log_2 c^{1+p(n)}$ 個のます目を要するだけであり，この値は $O(p(n))$ である．したがって，二つの ID と一つの整数から成るスタック枠の全体に要する領域量は $O(p(n))$ である．

D は，高々$O(p(n))$ 個のスタック枠を持つだけなので，使用される領域の最大量は $O(p^2(n))$ である．$p(n)$ が多項式であるならば，この領域量も多項式である．したがって，L は多項式領域有界の DTM によって受理されることが結論される．　▨

まとめとして，我々の計算量のクラスに関する知識に，多項式領域のクラスを追加した図を，図 11.5 に示す．

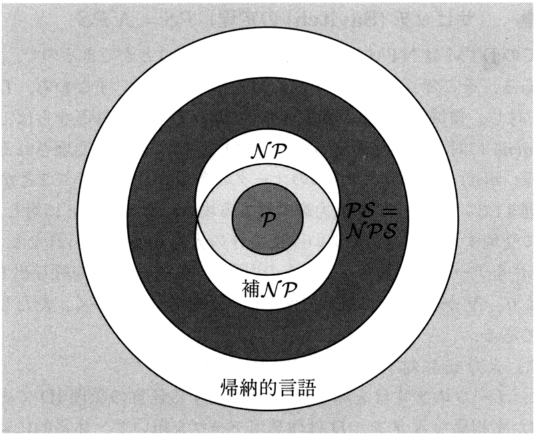


図 11.5　言語のクラスの間の既知の関係

11.3　$\mathcal{PS}$ に対して完全な問題

本節では,「限定記号付きブール式」と呼ばれる問題を紹介し，その問題が $\mathcal{PS}$ に対し完全であることを証明する．

11.3.1　*PS* 完全性

問題 P は，次を満たすとき，$\mathcal{PS}$ に対して完全である (*PS* 完全) と定義する．

(1)　P は $\mathcal{PS}$ に属す．

(2)　$\mathcal{PS}$ に属すすべての言語 L は，P に多項式時間還元可能である．

我々は，多項式領域について考えているのであって，時間について考えているのではないのだが，*PS* 完全性に対して要請される条件は *NP* 完全性に対して要請される条件と同じであることに注意されたい．すなわち，還元は多項式時間で行われなければならない．その理由は，もしある *PS* 完全問題が $\mathcal{P}$ に属すことがわかったときには $\mathcal{P} = \mathcal{PS}$ となって欲しいし，また，もしある *PS* 完全問題が $\mathcal{NP}$ に属すことがわかったときには $\mathcal{NP} = \mathcal{PS}$ が成り立って欲しいからである．仮に，還元に対する制約が多項式領域であったとすると，出力の大きさは入力の大きさに関して指数関数的になるかもしれず，したがって，次の定

理に述べるような結論を引き出すことはできなくなってしまう．しかし，我々は，多項式時間還元に焦点を絞るので，期待する関係を得ることができるのである．

定理 11.6 P は $\mathcal{PS}$ 完全問題であるとする．このとき，次が成り立つ．

a) P が $\mathcal{P}$ に属すならば，$\mathcal{P} = \mathcal{PS}$ である．

b) P が $\mathcal{NP}$ に属すならば，$\mathcal{NP} = \mathcal{PS}$ である．

証明 (a) を証明しよう．$\mathcal{PS}$ に属す任意の L に対し，L から P への多項式時間還元が存在する．この還元に要する時間を $q(n)$ とする．また，P は $\mathcal{P}$ に属すと仮定するので，P は多項式時間のアルゴリズムを持つ．このアルゴリズムの実行時間を $p(n)$ とする．

L に属すかどうかを判定したい文字列 w が与えられたとする．w を，x が P に属すことと w が L に属すこととが同値であるような文字列 x に変換する．変換に要する時間は $q(|w|)$ であるので，文字列 x は $q(|w|)$ より長くなることはない．x が P に属すかどうかのテストに要する時間は $p(|x|)$ であるが，これは $p(q(|w|))$ であり，$|w|$ に関する多項式である．以上より，L に対する多項式時間のアルゴリズムが存在することが結論される．

よって，$\mathcal{PS}$ に属すすべての言語 L は $\mathcal{P}$ に属す．一方，$\mathcal{P}$ が $\mathcal{PS}$ に含まれることは明らかなので，P が $\mathcal{P}$ に属すならば，$\mathcal{P} = \mathcal{PS}$ が成り立つことがわかる．

(b) に対する証明も，P が $\mathcal{NP}$ に属すことを仮定する点を除き，上とまったく同様であるので，読者に委ねることにする． ▨

11.3.2 限定記号付きブール式

$\mathcal{PS}$ に対して完全である一つの問題 P を紹介していく．それは，「限定記号付きブール式」，または，QBF と表記される問題である．まず，この問題を定義するために必要ないくつかの用語を学んでいくことにしよう．

雑ないい方をすると，限定記号付きブール式とは，ブール式に限定記号 $\forall$（すべての ... に対し）または $\exists$（ ... が存在する）を付加したものである．ブール式 $(\forall x)(E)$ は，E の中の x のすべての出現を 1（真）で置き換えて得られる式は真であり，かつ，E の中の x のすべての出現を 0（偽）で置き換えて得られる式も真であることを意味する．ブール式 $(\exists x)(E)$ は，E の中の x のすべての出現を 1（真）で置き換えて得られる式が真であるか，または，E の中の x のすべての出現を 0（偽）で置き換えて得られる式が真であることを意味する．(その両方の式が真である場合も含む.)

説明を簡単にするため，どの QBF についても同じ変数 x に対し**限定記号** (quantification，∀ または ∃) が 2 個以上適用されることはないと仮定する．この制限は本質的なものではなく，大雑把にいうと，プログラムの中で二つの異なる関数が同じ局所変数を使用することを許さない，ということに対応する[3)]．形式的にいうと，**限定記号付きブール式** (quantified boolean formula) は次のように定義される．

(1)　0（偽），1（真），および，任意の変数は，QBF である．

(2)　E と F が QBF であるとき，(E)，$\neg(E)$，$(E) \wedge (F)$，$(E) \vee (F)$ も QBF である．これらは，各々，括弧で囲まれた E，E の否定，E と F の AND，そして，E と F の OR を表す．演算子の間に通常の優先順位，すなわち，優先度の高い順に，NOT，AND，OR を適用し，不必要な括弧は省略することがある．また，AND と OR を表すのに，「算術式」のような記法を用いて，AND を連接（演算子省略），OR を + で表すことがある．すなわち，$(E) \wedge (F)$ の代わりに $(E)(F)$ を，また，$(E) \vee (F)$ の代わりに $(E) + (F)$ を用いる．

(3)　E が，変数 x に対する限定記号を含まない QBF であるとき，$(\forall x)(E)$ と $(\exists x)(E)$ も QBF である．x の**スコープ** (scope) は式 E であるという．直観的に言うと，x は E の内部だけで定義される，という意味である．プログラムの中で，変数のスコープはその変数が宣言された関数の内部である，というのと類似している．E を囲む括弧は，曖昧さがなければ，省略してよい．限定記号を囲む括弧は省略しない．ただし，括弧が多重になる煩雑さを避けるため，

$$(\forall x)((\exists y)((\forall z)(E)))$$

のような限定記号の連鎖を含む式を書くときには，連鎖の中に現れる限定記号ごとに一対の括弧を書くのではなく，E の周りに一対の括弧を書くだけにする．すなわち，上式は $(\forall x)(\exists y)(\forall z)(E)$ のように書く．

[3)] プログラムの場合も，また，限定記号付きブール式の場合も，同じ変数名が 2 回異なる使われ方をするときは，その一方に対して名前を付け替えることは常に可能である．プログラムの場合は，同じ局所変数の再使用を避けるべき理由は何もない．しかし，QBF の場合は，変数の再使用を仮定しない方が都合がよいときがある．

例 11.7 次式は QBF の例である．

$$(\forall x)((\exists y)(xy) + (\forall z)(\neg x + z)) \tag{11.1}$$

変数 x と y から始め，それらをまず AND で結び，次に，限定記号 $(\exists y)$ を適用して部分式 $(\exists y)(xy)$ を作る．同様に，ブール式 $\neg x + z$ を構成し，限定記号 $(\forall z)$ を適用して，部分式 $(\forall z)(\neg x + z)$ を作る．さらに，これらの二つの式を OR で結びつける．このとき，+(OR) の優先順位は最弱なので，括弧は必要ない．最後に，限定記号 $(\forall x)$ をこの式に適用して，上記の QBF が得られる．▨

11.3.3 限定記号付きブール式の評価

我々はまだ，QBF の持つ意味について形式的な定義を与えていない．しかし，$\forall$ を「すべての ... に対し」と読み，また，$\exists$ を「... が存在する」と読めば，直観的に意味を捉えることはできる．上の例で見た QBF は，すべての x に対し，(すなわち，$x = 0$ および $x = 1$ に対し，) x と y の両方が真であるような y が存在するか，または，すべての z に対し $\neg x + z$ が真である，ということを主張する．この文はたまたま値が真となる文の例になっている．なぜならば，$x = 1$ の場合は，$y = 1$ とすることにより xy を真にすることができるし，また，$x = 0$ の場合は，$\neg x + z$ は z の両方の値に対して真になるからである．

変数 x が x のある限定記号のスコープの中にあるとき，x のその出現は**限定** (bound) されているという．そうでないとき，x のその出現は**自由** (free) であるという．

例 11.8 式 (11.1) の QBF において，変数の出現はすべて限定されている．それらはすべて，その変数に対する限定記号のスコープの中にあるからである．たとえば，$(\exists y)(xy)$ の中で限定記号が付されている変数 y のスコープは式 xy である．したがって，そこでの y の出現は限定されている．式 xy における x の出現は，そのスコープが式全体である限定記号 $(\forall x)$ によって，限定されている．▨

自由変数を持たない QBF の値は 0 または 1（すなわち，偽または真）である．そのような QBF の値を，式の長さ n に関する帰納法によって計算することができる．

基礎：式の長さが 1 であるとき，それは，定数 0 または 1 以外ではあり得ない．変数のみの出現は自由な出現となってしまうからである．定数だけから成る式の値はそれ自身である．

帰納：自由変数を持たない，長さ $n > 1$ の式が与えられたとする．ここで，長さが n より短い式は，それが自由変数を持たない限り，評価する（値を求める）ことが可能であるとする．与えられた式がとり得る形として，6 個の可能な形がある．

(1) 式は，(E) の形である．このとき，E の長さは $n-2$ であり，値を評価することができて，0 または 1 の値が定まる．(E) の値は，それと同じ値である．

(2) 式は，$\neg E$ の形である．このとき，E の長さは $n-1$ であり，値を評価することができる．もし，$E=1$ であれば，$\neg E=0$ であり，$E=0$ であれば，$\neg E=1$ である．

(3) 式は，EF の形である．このとき，E も F も n より短いので，どちらも値を評価することができる．EF の値は，E と F の値がともに 1 であるとき 1 であり，少なくとも一方が 0 のとき 0 である．

(4) 式は，$E+F$ の形である．このとき，E も F も n より短いので，どちらも値を評価することができる．$E+F$ の値は，E と F の少なくとも一方の値が 1 であるとき 1 であり，ともに 0 のとき 0 である．

(5) 式が $(\forall x)(E)$ の形であるときは，まず，E における x のすべての出現を 0 で置き換えて式 E_0 を作る．同様に，E における x のすべての出現を 1 で置き換えて式 E_1 を作る．E_0 と E_1 はいずれも次の性質を満たすことに注意する．

 (a) 自由変数を持たない．なぜならば，E_0 または E_1 において自由変数の出現があるとすれば，それは x ではあり得ず，したがって，E においても同様に自由な変数が出現していることになってしまうからである．

 (b) 式の長さは $n-6$ であり，したがって，n より短い．

E_0 と E_1 を評価する．どちらの値も 1 であるならば，$(\forall x)(E)$ の値も 1 である．そうでない場合，$(\forall x)(E)$ の値は 0 である．この規則が，$(\forall x)$ の「すべての x に対し」という解釈を正しく反映していることに注意されたい．

(6) 与えられた式が $(\exists x)(E)$ の形であるときは，(5) と同様に作業を進め，E_0 と E_1 を構成し，それらを評価する．E_0 と E_1 の少なくとも一方の値が 1 であるならば，$(\exists x)(E)$ の値も 1 である．そうでない場合，$(\exists x)(E)$ の値は 0 である．この規則が，$(\exists x)(E)$ の「x が存在する」という解釈を正しく反映していることに注意されたい．

例 11.9 式 (11.1) の QBF 式を評価してみよう．この式は，$(\forall x)(E)$ の形をしている．そこで，まず，E_0 を評価する必要がある．E_0 は次のような式である．

$$(\exists y)(0y) + (\forall z)(\neg 0 + z) \tag{11.2}$$

この式の値は，OR で結ばれている二つの式 $(\exists y)(0y)$ と $(\forall z)(\neg 0 + z)$ の値に依存する．これらのうちの少なくとも一方の値が 1 であれば，E_0 の値も 1 である．そこで，まず，$(\exists y)(0y)$ を評価するため，部分式 $0y$ に $y = 0$ と $y = 1$ をそれぞれ代入し，それらのうちの少なくとも一方の値が 1 になるかどうかを確かめる．ところが，$0 \wedge 0$ と $0 \wedge 1$ の値はともに 0 であり，したがって，$(\exists y)(0y)$ の値は 0 である[4]．

幸いなことに，$(\forall z)(\neg 0 + z)$ の値は 1 である．そのことは，$z = 0$ と $z = 1$ の両方を，それぞれ，代入して見ればわかる．$\neg 0 = 1$ であるので，評価する必要のある二つの式は $1 \vee 0$ と $1 \vee 1$ となる．これらはいずれも値が 1 であるので，$(\forall z)(\neg 0 + z)$ の値も 1 であることがわかる．よって，式 (11.2) として表される式 E_0 について，その値は 1 であるという結論が得られた．

式 (11.1) において $x = 1$ を代入して得られる次の式 E_1 についても，やはり値が 1 になることを確かめなければいけない．

$$(\exists y)(1y) + (\forall z)(\neg 1 + z) \tag{11.3}$$

式 $(\exists y)(1y)$ の値は 1 である．そのことは，$y = 1$ を代入すればわかる．したがって，E_1，式 (11.3)，の値は 1 である．以上より，式の全体，すなわち，式 (11.1) の値は 1 であることが示された． ▨

---

[4] ここで，AND と OR に対し，もう一つの記法を用いていることに注意されたい．0 と 1 を含む式に対し連接（無記号）や + を用いると，複数桁の整数や算術的な加法のように見えてしまい具合が悪いからである．読者には，2 通りの記法のどちらも，同じ論理演算を表す記法として受け入れて頂けることと思う．

11.3.4 QBF 問題の $\mathcal{PS}$ 完全性

さて，いよいよ，**限定記号付きブール式問題** (quantified boolean formula problem) を定義する．それは次のような問題である．自由変数を持たない QBF が与えられたとき，それが値 1 を持つかどうかを判定せよ．我々は，この問題を QBF と呼ぶことにする．一方,「限定記号付きブール式」に対しても，これまでと同様，QBF という略称を用いる．QBF がどちらを意味するかは文脈から明らかであり，混乱は避けられるであろう．

我々は，QBF 問題は $\mathcal{PS}$ に対して完全であることを示す．証明は，定理 10.9 と定理 11.5 とでそれぞれ用いられたアイデアを組み合わせることによって得られる．定理 10.9 から借用するアイデアは，TM の計算を，あるます目がある時点である値を持つかどうかを示す論理変数の集まりとして表現する，というアイデアである．ただし，定理 10.9 がそうだったように，多項式時間を扱う場合には，考慮する必要のある変数は多項式個だけであり，したがって，TM が入力を受理するということを主張する式を多項式時間で生成することができた．これに対し，多項式領域を限界とする計算を扱う場合には，計算の中に現れる ID の個数は，入力のサイズに関して指数関数的になるかもしれず，そのような場合，多項式時間では，計算が正しいことを主張するブール式を書き下すことができない．幸い，我々には，我々が言う必要のあることを表現できるだけの，前よりも強力な言語が与えられている．すなわち，限定記号を利用することにより，我々は，多項式領域有界の TM が入力を受理するということを主張する，多項式の長さの QBF を書くことが可能になる．

定理 11.5 からは，一つの ID から，ある多数回の動作を経て，別の ID に移る，という考えを表現するための「再帰的二重法」のアイデアを借用する．すなわち，ID I は m 回の動作の後 ID J に移る，ということを言う代わり，我々は，ある ID K が存在し，I は $m/2$ 回の動作ののち K に移り，さらに，K は $m/2$ 回の動作ののち J に移る，と言う．限定記号付きブール式から成る言語は，これらのことを，たとえ，m が入力の長さに関して指数関数的な値であったとしても，多項式の長さの式でいうことを可能にする．

$\mathcal{PS}$ に属す任意の言語は QBF に多項式時間還元可能である，ということを証明するとともに，QBF は $\mathcal{PS}$ に属す，ということも示す必要がある．$\mathcal{PS}$ 完

全性の証明のこの部分でさえ，多少の工夫が要求される．そこで，我々は，この部分だけを一つの独立した定理として取り上げることにする．

定理 11.10 QBF は $\mathcal{PS}$ に属す．

証明 11.3.3 項で，我々は，QBF F を評価するための再帰的な処理について論じた．このとき説明したアルゴリズムをスタックを使って実現することができる．スタックは，定理 11.5 の証明のときと同様に，テューリング機械のテープ上に貯えればよい．F の長さを n としよう．このとき，F に対し，F 自身と，我々の作業の対象となる F の部分式に関する記述のための領域とを含む，長さ $O(n)$ のレコードを作る．F の形として 6 種類のものがあり得るが，そのうちの二つの例について考察すれば，評価の処理の方法が明らかになるであろう．

(1) $F = F_1 + F_2$ とする．このとき，次の処理を行う．

- (a) F に対するレコードの右に，F_1 に対するレコードのための領域を用意し，そこに F_1 を置く．
- (b) 再帰的に F_1 を評価する．
- (c) F_1 の値が 1 である場合，F の値として 1 を返す．
- (d) F_1 の値が 0 である場合は，F_1 に対するレコードを F_2 に対するレコードで置き換え，再帰的に F_2 を評価する．
- (e) F_2 の値を，それが何であっても，F の値として返す．

(2) $F = (\exists x)(E)$ とする．このとき，次の処理を行う．

- (a) x のすべての出現に対し 0 を代入することにより，式 E_0 を作る．F に対するレコードの右に，E_0 に対するレコードのための領域を用意し，そこに E_0 を置く．
- (b) 再帰的に E_0 を評価する．
- (c) E_0 の値が 1 である場合，F の値として 1 を返す．
- (d) E_0 の値が 0 である場合は，E の中の x のすべての出現に対し 1 を代入することにより，式 E_1 を作る．
- (e) E_0 に対するレコードを E_1 に対するレコードで置き換え，再帰的に E_1 を評価する．
- (f) E_1 の値を，それが何であっても，F の値として返す．

F がこれら以外の四つの形，すなわち，F_1F_2，$\neg E$，(E)，または，$(\forall x)(E)$ のいずれかの形である場合も，F を評価する方法は上と同様であるが，詳細は読者にお任せする．基礎の場合，すなわち，F が定数である場合は，その定数をそのまま返せばよい．このとき，新しいレコードがテープ上に作られることはない．

いずれの場合にも，長さ m の式に対するレコードの右方に作られるレコードは，長

さが m より短い式に対するレコードであることに注意する．また，二つの異なる部分式を評価しなければならなくなることが多いが，そのようなときでも，一度に一つずつ評価を行えばよいことに注意されたい．したがって，上記の (1) の場合，F_1，あるいは，その任意の部分式に対するレコードと，F_2，あるいは，その任意の部分式とに対するレコードが，同時にテープ上にのる事態は決して生じない．上記 (2) の場合の E_0 と E_1 についても，事情はまったく同じである．

よって，長さ n の式から出発するとき，n 個を超える個数のレコードがスタック上に同時に現れることはない．また，各々のレコードの長さは $O(n)$ である．したがって，テープ全体の使用量が $O(n^2)$ より大きくなることはない．こうして，QBF を受理する多項式領域有界 TM に対する構成法が得られた．この TM の領域量は n の 2 乗のオーダーである．なお，このアルゴリズムは，多くの場合，n に関する指数関数の時間を必要とすることに注意されたい．したがって，このアルゴリズムは多項式時間有界ではない．▨

さて，次に，$\mathcal{PS}$ に属す任意の言語 L から QBF 問題への還元の方法について考えることにしよう．定理 10.9 のときと同様に，i 番目の ID の j 番目の位置には記号 A があるということを主張するために，命題変数 y_{ijA} を用いることにしたい．しかし，ID の個数は指数関数個あるため，長さ n の入力 w を受け取るとき，n に関する多項式の時間ではこれらの変数を書き下すことさえできない．そこで，そうする代わりに，我々は，限定記号を利用して，同じ変数の集合で多くの異なる ID を表現できるようにする．このアイデアは，次に記す証明の中で詳しく説明される．

定理 11.11　QBF 問題は PS 完全である．

証明　L を $\mathcal{PS}$ に属す任意の言語とする．L は，長さ n の入力に対して高々$p(n)$ の領域を使用する決定性 TM M によって受理されるものとする．定理 11.3 より，M は，長さ n の入力を受理するならば，$c^{1+p(n)}$ 回以下の動作の後に受理する，という性質を満たす定数 c が存在する．以下，我々は，長さ n の入力 w を受け取り，w から，自由変数を持たない QBF E で，E の値が 1 であることと w が $L(M)$ に属すこととが同値であるようなものを，多項式時間で構成する方法を説明していく．

E を記述するにあたり，多項式個の**変数** ID (variable ID) を導入すると都合がよい．変数 ID とは，表現されるべき ID の j 番目の位置には記号 A があることを主張する，変数 y_{jA} の集合である．j は，0 以上 $p(n)$ 以下の範囲を動く．記号 A は，テープ記号か，または，M の状態である．したがって，変数 ID の中の命題変数の個数は，n に関する多項式である．相異なる変数 ID の間では，命題変数はすべて異なるもの

を用いるものと仮定する．すなわち，どの命題変数も二つの異なる変数 ID に属すことはないとする．多項式個の変数 ID だけを扱う限り，命題変数の総数は多項式個である．

変数 ID I に対し，$(\exists I)$ という記法を導入すると便利である．変数 ID I に含まれる命題変数の全体を $x_1, x_2, \ldots, x_m$ とするとき，この限定記号は，$(\exists x_1)(\exists x_2)\cdots(\exists x_m)$ の短縮形であるとする．同様に，$(\forall I)$ は，I に含まれるすべての命題変数に限定記号 $\forall$ を適用したものの短縮形であるとする．

入力 w に対し，我々が構成する QBF は次のような形の式である．

$$(\exists I_0)(\exists I_f)(S \wedge N \wedge F)$$

ただし，

(1) I_0 と I_f は，各々，初期 ID と受理 ID を表す変数 ID である．

(2) S は,「計算は正しく始まる」こと，すなわち，I_0 は入力 w に対する M の初期 ID であること，を主張する式である．

(3) N は,「動作は正しい」こと，すなわち，M は I_0 を I_f に移すこと，を主張する式である．

(4) F は,「計算は正しく終わる」こと，すなわち，I_f は受理 ID であること，を主張する式である．

式全体として見ると自由変数は一つも含まれないが，部分式では必ずしもそうではないことに注意されたい．すなわち，I_0 の変数は S の中に自由変数として現れるし，I_f の変数は F の中に自由変数として現れる．さらに，それらの両方のグループの変数は N の中に自由変数として現れる．

計算は正しく始まる　S はリテラルの論理積（AND）である．各リテラルは，I_0 の変数の一つまたは，その否定，である．S は，入力 w を持つ初期 ID の j 番目の位置の記号が A であるとき，リテラル y_{jA} を含む．そうでないとき，S はリテラル $\overline{y_{jA}}$ を含む．すなわち，$w = a_1a_2\cdots a_n$ のとき，y_{0q_0}, y_{1a_1}, $y_{2a_2}, \ldots$, y_{na_n}，および，$j = n+1, n+2, \cdots, p(n)$ に対するすべての y_{jB} は否定演算子を伴わない形で現れ，I_0 に属すその他のすべての変数は否定演算子付きで現れる．なお，ここで，q_0 は M の初期状態とし，B は空白記号であるとする．

計算は正しく終わる　I_f が受理 ID であるためには，I_f は受理状態を含まなければならない．したがって，F を，I_f の命題変数から選ばれた変数 y_{jA} で，A が受理状態であるようなものの全体に対する論理和（OR）として書く．位置 j は任意である．

次に進む動作は正しい　式 N は再帰的に構成される．その方法は，それまでに構成されている式に，$O(p(n))$ 個の記号を付け加えるだけで，そして，(さらに重要なことだが,) 式を書き下すのに $O(p(n))$ 時間しか使わずに，考慮される動作数を 2 倍に

N_{2i} のうまくいかない構成の例

N_i から N_{2i} を構成する方法に関する我々の第一感は，分割統治法を直接的に利用することであるかもしれない．すなわち，$2i$ 以下の動作数で $I \overset{*}{\vdash} J$ となるならば，ともに i 以下の動作数で $I \overset{*}{\vdash} K$ および $K \overset{*}{\vdash} J$ となるような ID K が存在しなければならない．しかしながら，この考え方を式の形でそのまま書き下すとすると，たとえば，$N_{2i}(I,J) = (\exists K)(N_i(I,K) \wedge N_i(K,J))$ と書くとすると，i を 2 倍するたびに式の長さを 2 倍することになってしまう．M のすべての可能な計算を表すためには，i は n に関して指数のオーダでなければならないので，N を書き下すのに時間がかかり過ぎてしまうし，N の長さは指数になってしまい具合が悪い．

するという方法である．変数 ID I, J に対し，$I = J$ という略記法を採用すると便利である．これは，I と J の対応する変数の各々を等号でつないだ式の論理積（AND）を意味するものとする．すなわち，I が変数 y_{jA} から成り，J が変数 z_{jA} から成るとき，$I = J$ は，$(y_{jA}z_{jA} + (\overline{y_{jA}})(\overline{z_{jA}}))$ という式を AND で結んだ式を表す．ここで，j は 0 以上，$p(n)$ 以下の範囲を動き，A は，M の任意のテープ記号，または，状態である．

さて，次に，$i = 1, 2, 4, 8, \cdots$ に対し，i 回以下の動作で $I \overset{*}{\vdash} J$ となることを意味する式 $N_i(I,J)$ を構成する．これらの式では，変数 ID I と J に含まれる命題変数だけが自由変数であり，その他のすべての命題変数は束縛変数である．

基礎：$i = 1$ のとき，$N_i(I,J)$ は，$I = J$ または $I \vdash J$ を主張する．条件 $I = J$ の表し方は，いま，上で論じたばかりである．条件 $I \vdash J$ については，定理 10.9 の証明の中の「次に進む動作は正しい」の部分で示された議論を参照していただきたい．そこでは，ID が一つ前の ID から正しく導かれることを主張する問題，まさにここで必要とされるのと同じ問題が扱われていた．式 N_i は，これらの二つの式を論理和（OR）で結んだものである．N_i は $O(p(n))$ 時間で書くことができることに注意されたい．

帰納：$N_{2i}(I,J)$ を N_i から構成する．上の枠内の「N_{2i} のうまくいかない構成の例」で指摘したように，N_{2i} を構成するために N_i のコピーを二つ用いる直接的な方法では，我々が必要とする時間量や領域量は得られない．N_{2i} を書く正しい方法は，式の中で N_i のコピーを一つだけ用いる方法である．同じ式に，引数 (I,K) と (K,J) の両方を順に渡すのである．すなわち，$N_{2i}(I,J)$ は，部分式 $N_i(P,Q)$ を一つしか使わない．我々は，$N_{2i}(I,J)$ を，すべての ID P と Q に対し，次の (1)，または，(2) を満たすような ID K が存在することを主張する式として書く．

(1)　$(P,Q) \neq (I,K)$ かつ $(P,Q) \neq (K,J)$，または，

(2)　$N_i(P,Q)$ は真である．

言い方をかえると，$N_i(I,K)$ と $N_i(K,J)$ は真であるが，それ以外の $N_i(P,Q)$ についてはその真偽を問わない，ということである．$N_{2i}(I,J)$ に対する QBF を次に示す．

$$N_{2i}(I,J) = (\exists K)(\forall P)(\forall Q)(N_i(P,Q) \vee (\neg(I=P \wedge K=Q) \wedge \neg(K=P \wedge J=Q)))$$

N_{2i} を書くために要する時間は，N_i を書くために要する時間に，$O(p(n))$ の時間を追加した時間であることに注意されたい．

N の構成を完成させるため，2 のベキ乗であり，かつ，$c^{1+p(n)}$ 以上であるような最小の整数 m に対し，N_m を構成しなければならない．$c^{1+p(n)}$ は，TM M が長さ n の入力 w を受理するまでに行うことの可能な動作数の上界である．上記の再帰的ステップを適用する回数は，$\log_2(c^{1+p(n)})$，すなわち，$O(p(n))$ である．再帰的ステップを 1 回適用するために要する時間は $O(p(n))$ であるので，N は時間 $O(p^2(n))$ で構成することができることが結論される．

定理 11.11 の証明の結論　これまでの議論で，入力 w を QBF

$$(\exists I_0)(\exists I_f)(S \wedge N \wedge F)$$

に $|w|$ に関する多項式の時間で変換する方法を示した．我々はまた，S，N，F の各式が真になるための必要十分条件は，式の中の自由変数が，入力 w に対する M の計算の初期 ID と受理 ID である ID I_0 と I_f を表し，かつ，$I_0 \overset{*}{\vdash} I_f$ が成り立つときであることの理由も示した．すなわち，この QBF が値 1 を持つための必要十分条件は M が w を受理することである． ▨

11.3.5 11.3 節の演習問題

♣ **問 11.3.1**　次の各々の場合を考察することにより，定理 11.10 の証明を完成せよ．

a) $F = F_1F_2$

b) $F = (\forall x)(E)$

c) $F = \neg(E)$

d) $F = (E)$

♣*!! **問 11.3.2**　次の問題は PS 完全であることを示せ．正則表現 E が与えられたとき，E は Σ^* と同等であるかどうかを判定せよ．ただし，Σ は，E の中に現れる記号全体の集合であるとする．**ヒント**：QBF をこの問題に還元しようとするよりも，PS に属す任意の言語はこの問題に還元可能であるということを示す方がやさしいかもしれない．任意の多項式領域有界 TM M に対し，M に対する入力 w を受け取り，w を受理に導く M の ID 列ではないすべての文字列を生成する

正則表現を，多項式時間で構成する方法を示せ．

♣!! **問 11.3.3** **シャノンのスイッチング・ゲーム** (Shannon Switching Game) とは，次のようなゲームである．2個の定頂点 s と t を持つグラフ G が与えられたとする．2人のプレーヤーがいる．1人を SHORT と呼び，もう1人を CUT と呼ぶ．SHORT から先に始め，2人のプレーヤーは交互に，s と t 以外で，まだどちらのプレーヤーにも帰属していない G の頂点を選ぶ．選ばれた頂点は，以後，選んだプレーヤーに帰属する頂点となる．SHORT が勝つのは，選んだ頂点に s と t を含めた集合が，s から t への路を形成する頂点の集合を含むようにできたときである．一方，CUT が勝つのは，すべての頂点が選ばれたのち，SHORT が s から t への路を選ぶことができなかったときである．次の問題は $\mathcal{PS}$ 完全であることを示せ．グラフ G が与えられたとき，そのグラフに関して，CUT の選択のしかたによらず，SHORT には必ず勝つ方法があるかどうか（すなわち，先手必勝であるかどうか）を判定せよ．

11.4 乱数を利用して定義される言語のクラス

話題を転じ，本節では，乱数に関わって定義される言語の二つのクラスを考察する．対象となるのは，計算の過程で乱数を使用する能力を備えたテューリング機械によって定義される言語のクラスである．読者は，恐らく，通常のプログラミング言語で書かれたアルゴリズムで，ある種の有用な目的のために乱数生成ルーチンを使用するものに馴染みがあるであろう．正確にいえば，一見したところ「乱数」，または，予知不能な数のように見える数を返してくる関数 `rand( )`，あるいは，それと類似の名前を持つ関数があるが，それらは，生成される数の列の中に一定の「パターン」を見い出すことが大変難しいとはいえ，実は，単に特定のアルゴリズムを実行するものにすぎず，真の乱数を生成するものではない．(実用されている例ではないが,) そのような関数の簡単な例に次のようなものがある．一つ前に生成された整数に対し，それを2乗し，得られた積のビット表現の中央部分を取り出し，それを次の「乱数」とする．通常使用する関数はもう少し複雑であるが，このように，機械的な操作を経て生成される数を**擬似乱数** (pseudo-random number) と呼ぶ．

本節では，乱数の生成とアルゴリズムにおけるそれらの数の使用をモデル化して，テューリング機械の一つの変型を定義する．次に，このような乱数性と多項式時間有界性とを異なる意味で用いることにより，二つの言語のクラス $\mathcal{RP}$

と $\mathcal{ZPP}$ を定義する．興味深いことに，これらのクラスはいずれも $\mathcal{P}$ に属さない言語をほんの僅かしか含まないクラスのように見える．しかし，そのほんの僅かの差が重要な差なのである．とくに，11.5 節で，我々は，コンピュータシステムの安全性に関して最も本質的であると思われる事柄のうちのいくつかは，実は，これらのクラスと $\mathcal{P}$ および $\mathcal{NP}$ との間の関係に由来するものであることを見ていく．

11.4.1　クイックソート：乱数アルゴリズムの例

読者は，多分,「クイック・ソート」と呼ばれるソーティングのアルゴリズムをご存じだろうと思う．アルゴリズムの本質は次の点にある．ソートされるべき要素のリスト $a_1, a_2, \ldots, a_n$ が与えられたとき，要素の一つ，たとえば，a_1 を選択し，a_1 以下の要素のグループと a_1 より大きい要素のグループにリストを分割する．選択された要素は，**軸** (pivot) の要素と呼ばれる．データの表現やしまい方を注意深く行えば，長さ n のリストを二つのリストに分割する作業は $O(n)$ の時間で行うことができる．さらに，再帰的に，小さい要素（軸の要素以下の要素）のリストのソートと，大きい要素（軸の要素より大きい要素）のリストのソートを，互いに独立に行うことができる．その結果，n 個の要素すべてのソートされたリストが得られる．

運がよければ，軸の要素として，ソートすべきリストの中の中央に位置する要素が選ばれ，したがって，二つの部分リストは，各々，およそ $n/2$ の長さとなる．再帰的な処理の各局面で，いつも運がよければ，再帰法をおよそ $\log_2 n$ 個のレベルの深さだけ繰り返したのちに，長さ 1 のリストを得ることになる．長さ 1 のリストはすでにソートされているリストであるので，そこで再帰法の処理は終了する．全体の仕事の深さのレベルは $\log_2 n$ 個であり，各レベルごとに $O(n)$ の仕事が行われるので，処理全体に要する時間は $O(n \log n)$ となる．

しかし，我々はいつも運がよいとは限らない．たとえば，ソートすべきリストが，実は，すでにソートされたリストである，ということも起こり得る．そのとき，各リストの先頭のリストを軸の要素にとるとすると，リストは，小さい要素の部分リストには 1 個の要素だけが入り，残りはすべて大きい要素のリストに入るという形に分割されるだけである．そのような場合，クイック・ソートは選択ソートとほぼ同様の処理を行うこととなり，n 個の要素をソートする

のに，n^2 に比例する時間がかかることになる．

このような考えから，クイック・ソートのよい実現法では，リストの中の特定の位置の要素を，機械的に，軸の要素にすることはしない．そうではなく，軸の要素は，リストに含まれるすべての要素の中からランダムに選ばれる．すなわち，n 個の要素の各々について，その要素が軸の要素として選ばれる確率は $1/n$ である．ここでは証明を記さないが[5)]，このような乱数の利用を含むクイック・ソートの平均実行時間は $O(n \log n)$ であることを示すことができる．ただし，極めて微少な確率ではあるが，軸の要素としていつも最大の要素，または，最小の要素が選ばれることがあり得るので，クイック・ソートの最悪の場合の実行時間は，やはり，$O(n^2)$ であることに変わりはない．それでも，クイック・ソートは，各種の応用の場で広く採用されている．(たとえば，UNIX のソーティングの命令ではクイック・ソートが用いられている.) その理由は，クイック・ソートの平均実行時間が，他の方法と比べたとき，たとえば，最悪の場合の実行時間が $O(n \log n)$ であるような方法と比べた場合でさえ，大変優れているというところにある．

11.4.2 乱数を利用するテューリング機械のモデル

テューリング機械に付加するランダムな選択を行う能力を抽象的に表現するため，図 11.6 に示すような多テープ TM の変型を定める．これは，乱数生成ルーチンを何回も呼び出すプログラムに類似のモデルである．第 1 テープには，通常の多テープ TM の場合と同様に，入力列を置く．第 2 テープの各ます目にも，やはり，予め，空白以外の文字を置いておく．実際，原理的には，テープ上のます目の全体が 0 または 1 で埋められていると考えてよい．各々のます目に 0 が置かれる確率も，また，1 が置かれる確率も，いずれも 1/2 であり，それらはランダムに，かつ，互いに独立に選ばれる．この第 2 テープを**乱数テープ** (random tape) と呼ぶ．第 3 テープ，および，それ以後のテープを必要に応じて用いてよい．それらは，最初はすべて空白であり，TM により「計算用テープ」として使用される．このような TM のモデルを**ランダム・テューリング機**

5) クイック・ソートの平均実行時間の解析と証明については，次の文献の中に記されている．D. E. Knuth 著，*The Art of Computer Programming, Vol. III: Sorting and Searching*, Addison-Wesley, 1973.

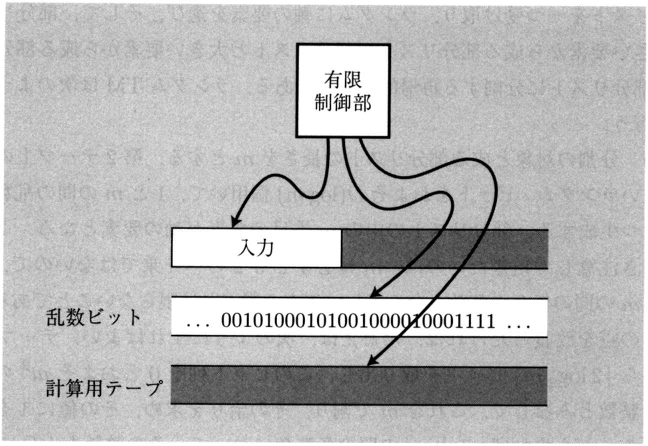


図 11.6 ランダムに「生成された」数を利用する能力を持つテューリング機械

械 (randomized Turing machine) と呼ぶ.

ランダム TM を，その無限の長さのテープの全体をランダムに選ばれた 0 と 1 とで覆うように初期化する，と想像するのは現実的でないかも知れない．そこで，第 2 テープは，最初，空白のみであるとする別の設定のしかたもある．その場合は，第 2 テープのヘッドが空白を読むと，TM の内部で「コイン投げ」が行われ，ランダム TM は直ちに 0 または 1 をテープ上のヘッドが見ているます目に記す．一度記された 0 または 1 は，その後，書き換えられることなくテープのその位置にずっと残る．このような定め方をすると，ランダム TM の仕事を始める前に，予め —— 無限の仕事をする必要がないどころか —— 何の準備作業もしておく必要はない．それでいながら，第 2 テープはランダムな 0 と 1 とで覆われているかのように見える．ランダム TM の第 2 テープのヘッドが実際にます目を見ようとするときにはいつも，これらのランダム・ビットが出現するからである.

例 11.12 クイック・ソートの乱数版をランダム TM 上に実現することができる．処理の途中で各部分リストは，入力テープ上に，両端にマーカーによる境界の印が付されて，連続して配置されていると仮定する．重要なステップは,

部分リストを一つ受け取り，ランダムに軸の要素を選び，そして，部分リストを小さい要素から成る部分リストの部分リストと大きい要素から成る部分リストの部分リストに分割する再帰的な処理である．ランダム TM は次のような動作を行う．

(1) 分割の対象と成る部分リストの長さを m とする．第2テープ上の新しいランダム・ビットをおよそ $O(\log m)$ 個用いて，1と m の間の乱数を一つ生成する．部分リストの中の m 番目の要素が軸の要素となる．このとき注意して頂きたいのは，m は必ずしも2のベキ乗ではないので，1と m の間の任意の整数を全く等しい確率で選ぶとは限らないことである．この点を改良したければ，たとえば，次のようにすればよい．テープ2から $\lceil 2\log_2 m \rceil$ ビットを取り出し，このビット列を0とおよそ m^2 の間の整数とみなして，これを m で割り，その余りを求め，その値に1を加える．こうすれば，1と m の間の各整数について，その整数を十分に $1/m$ に近い確率で選ぶことができ，クイック・ソートを正しく機能させることができる．

(2) 軸の要素をテープ3に置く．

(3) テープ1上の境界の印で囲まれている部分リストの要素を順に見ていき，軸の要素以下の大きさのものをテープ4にコピーしていく．

(4) 再び，テープ1上の部分リストの要素を順に見ていき，軸の要素より大きいものをテープ5にコピーしていく．

(5) テープ1の境界で囲まれた部分リストが配置されていた場所に，まず，テープ4の内容をコピーし，次に，テープ5の内容をコピーする．二つのリストの間にマーカーを置く．

(6) 上で作られた部分リストの部分リストの各々について，2個以上の要素を含む場合は，同じアルゴリズムを用いて，再帰的にソートを行う．

クイック・ソートのこの実現は，計算の装置として，通常のコンピュータの代わりに多テープ・テューリング機械を用いているにも関わらず，平均計算時間は $O(n\log n)$ であることに注意されたい．ただし，この例を取り上げた目的は，実行時間を示すところにあるのではない．テューリング機械にランダムな行動を強いるための，第2テープのランダム・ビットの利用法を紹介すること

が主眼である. ▨

11.4.3 ランダム・テューリング機械の言語

我々がこれまで慣れ親しんできた状況は，すべてのテューリング機械（あるいは，FA や PDA）はある言語を受理する，という状況である．受理する言語は空集合であるかもしれないし，入力アルファベット上のすべての文字列の集合であるかもしれないが，それでも，言語を受理することに変わりはない．これに対し，ランダム・テューリング機械を扱う際は，TM が入力を受理するということの意味について，もう少し注意深く考える必要がある．ランダム TM の場合は，どんな言語もまったく受理しないということも起こり得るのである．問題は，ランダム TM M が入力 w に対して取る行動について考えるとき，ランダム・テープに対するすべての可能な内容を考慮して，M の行動を考える必要がある，という点にある．M があるランダム列に対しては入力を受理するが，別のランダム列に対しては入力を受理しない，ということは，当たり前のこととして起こり得るのである．実際，ランダム TM が何らかの仕事を決定性 TM よりも効率よく行おうとするとき，ランダム・テープ上の相異なる内容が TM の相異なる行動を導く，ということが本質的に重要なことである[6]．

ランダム TM を，通常の TM の場合と同じように，受理状態に入ることにより入力を受理する方式の TM であると考えると，ランダム TM M に対する各々の入力 w は，ある確率で受理されることになる．その確率は，ランダム・テープの内容全体に対して，受理に導くことが可能な内容の比率である．テープの内容は全部で無限個あるので，この確率の計算にあたっては，多少の注意深さが必要となる．しかし，受理に導く動作の系列は，いずれも，ランダム・テープのうちの有限の部分しか見ることがない．したがって，その部分に現れ得る列の出現の確率は，有限の値，すなわち，2^{-m} である．ここで，ランダム・テープ上のます目で，計算途中でヘッドに見られることがあり，その結果，TM の動作に少なくとも 1 回は影響を与えるます目の個数を m としている．次の例に，非常に簡単な場合であるが，計算の方法を示す．

---

[6] 例 11.12 で説明したランダム TM は言語を認識する TM ではないことに注意されたい．それは，言語の認識ではなく，入力に関する変換を行うものである．また，ランダム・テープの内容に依存するのは，変換に要する計算時間であり，出力結果ではない．

例 11.13 ランダム TM M は図 11.7 に示すような遷移関数を持つとする．M が使用するテープは，入力テープとランダム・テープだけである．M は非常に単純な動きしかしない．すなわち，どちらのテープ上の記号も一切書き換えず，テープ・ヘッドは右方に動かす（方向 R）か，または，留めておく（方向 S）だけである．我々はランダム TM の遷移に関する形式的な記法を定義していないが，それでも，図 11.7 の中の項目は十分理解できるであろう．各行は一つの状態に対応し，各列は記号の対 XY に対応する．ただし，X は入力テープのヘッドが見る記号であり，Y はランダム・テープのヘッドが見る記号である．表の中の項目 $qUVDE$ が意味するのは次の事柄である．TM は状態 q に入り，入力テープ上に U を書き，ランダム・テープに V を書き，さらに，入力テープのヘッドを方向 D に動かし，ランダム・テープのヘッドを方向 E に動かす．

ここに，0 と 1 とから成る入力列 w に対して M がとる動作をまとめておこう．M は，初期状態 Q_0 にいて，最初のランダム・ビットを見る．ランダム・ビットが 0 であるか 1 であるかにより，w に関する二つのテストのうちの一方を実行する．

ランダム・ビットが 0 のとき，M は，w がただ 1 種類の記号，0 または 1，だけから成る列であるかどうかを調べていく．この場合，M はそれ以上ランダム・ビットを見ることはなく，第 2 テープのヘッドは同じ位置に留めたままにしておく．w の第 1 ビットが 0 であるならば，M は状態 q_1 に行く．その状態にいて，M は 0 を読み飛ばして右方へヘッドを移動する．もし 1 を見たら，M は w を受理しない．M が状態 q_1 にとどまったままで，入力テープ上でヘッドを移動していくうちに空白を見たら，状態 q_4 に移る．q_4 は受理状態である．同様に，ランダム・ビットが 0 であり，w の第 1 ビットが 1 であるならば，M は

	00	01	10	11	$B0$	$B1$
$\rightarrow q_0$	$q_1 00RS$	$q_3 01SR$	$q_2 10RS$	$q_3 11SR$		
q_1	$q_1 00RS$				$q_4 B0SS$	
q_2			$q_2 10RS$		$q_4 B0SS$	
q_3	$q_3 00RR$			$q_3 11RR$	$q_4 B0SS$	$q_4 B1SS$
$*q_4$						

図 11.7　ランダム・テューリング機械の遷移関数

状態 q_2 に行く．その状態において，w の残りのビットがすべて 1 であるかどうかを調べる．残りがすべて 1 であれば，w を受理する．

次に，最初のランダム・ビットが 1 のとき，M がどのように行動するかを考えてみよう．M は，w と 2 番目以降のランダム・ビット列とを比較し，それらが同じ列であるとき，w を受理する．そのため，状態 q_0 において第 2 テープ上に 1 を見たら，M は状態 q_3 に移る．この動作を行うとき，M はランダム・テープのヘッドを右方に移すことに注意されたい．したがって，ヘッドは新しいランダム・ビットを見ることになる．一方，入力テープのヘッドは停めたままにしておくので，w の全体がランダム・ビット列と比較されることになる．状態 q_3 で，M は，両方のテープ・ヘッドを右方に動かしながら二つのテープの内容を照合していく．途中のどこかで一致しない個所が見つかったら，M は w を受理しない．一致したままで入力テープのヘッドが空白を見ることになったら，M は w を受理する．

さて，個々の入力について，受理の確率を計算してみよう．最初に，均一な入力，すなわち，ある $i \geq 1$ に対する 0^i のようにただ一つだけの文字から成る入力の場合を考える．確率 1/2 で，最初のランダム・ビットは 0 である．そして，その場合，均一性のテストは成功するので，0^i のような入力は必ず受理される．一方，同じ確率 1/2 で，最初のランダム・ビットは 1 である．このとき，0^i は，ランダム・ビットの 2 番目から $i+1$ 番目までがすべて 0 のとき，またそのときに限り，受理される．このようなことが起こる確率は 2^{-i} である．以上をまとめると，0^i が受理される確率は

$$\frac{1}{2} + \frac{1}{2}2^{-i} = \frac{1}{2} + 2^{-(i+1)}$$

であることがわかる．

今度は，均一でない入力，すなわち，00101 のように 0 と 1 の両方が含まれる入力の場合を考えてみよう．このような入力は，最初のランダム・ビットが 0 のときは，決して受理されることがない．最初のランダム・ビットが 1 のときは，入力の長さを i とするとき，受理の確率は 2^{-i} である．したがって，長さ i の均一でない入力に対する受理の確率は，$2^{-(i+1)}$ となる．たとえば，00101 の受理の確率は 1/64 である．　▨

我々は，次のような結論を得る．すなわち，任意に与えられたランダム TM による，任意に与えられた入力列に対する受理の確率を計算することが可能である．このとき，入力列が言語に属すか否かは，ランダム TM の言語を定める際に「所属性」をどのように定義するか，という点に依存する．我々は，次節で，このことに関して二つの異なる定義を与える．各々の定義から，相異なる「言語のクラス」が導かれる．

11.4.4 言語のクラス $\mathcal{RP}$

最初に考察する言語のクラスは $\mathcal{RP}$ と呼ばれるクラスである．この名前は「ランダム多項式」(random polynomial) に由来するものであり，その本質は次の点にある．言語 L が $\mathcal{RP}$ に属すためには，L はランダム TM M によって，次に記す意味で，受理されなければならない．

(1) w が L に属さないとき，M が w を受理する確率は 0 である．

(2) w が L に属すとき，M が w を受理する確率は少なくとも 1/2 である．

(3) 次を満たす多項式 $p(n)$ が存在する．入力 w の長さを n とするとき，ランダム・テープの内容によらず，M のすべての計算の流れは高々 $p(n)$ ステップの後に停止する．

$\mathcal{RP}$ の上記の定義の中には，二つの独立な事柄が提示されていることに注意されたい．項目 (1) と (2) により，特殊な型のランダム・テューリング機械が定義される．このような方式は，**モンテカルロ・アルゴリズム** (Monte-Carlo algorithm) と呼ばれることがある．すなわち，計算時間は一切問わないが，入

非決定性とランダム性

ランダム TM と非決定性 TM の間には，見かけ上，類似する点がある．NTM の非決定性の選択は，ランダム・ビットを持つテープによって支配されると考えることもできる．すなわち，NTM が次の動作を選択しなければならないときには，いつでも，ランダム・テープの内容を見て決めることにする．したがって，選択可能な動作の各々はいつも等しい確率で選択されることになる．しかし，NTM をこのように解釈するとき，この受理の規則は，$\mathcal{RP}$ に対する受理の規則とは明らかに異なる．NTM の場合，入力は，受理の確率が 0 であるときは受理されない．一方，受理の確率が 0 より大きな値のときは，たとえ，それがどんなに小さい値であったとしても，入力は受理される．

力を確率 0 で受理するか，または，少なくとも 1/2 の確率で受理する，その中間の場合は存在しない，というとき，ランダム TM は「モンテカルロ」型であるという．項目 (3) は，計算時間について述べているだけであり，TM が「モンテカルロ」型であるかどうかということとは独立な事柄である．

例 11.14　例 11.13 に示したランダム TM を考えてみよう．このランダム TM は，ランダム・テープの内容によらず，計算時間が $O(n)$ であるので，条件 (3) は確かに満たす．しかし，この TM は，$\mathcal{RP}$ の定義の中で要求されている意味では，いかなる言語も受理することはない．その理由は次の通りである．000 のような均一な入力は少なくとも 1/2 の確率で受理され，したがって，項目 (2) を満たすが，それ以外の入力で，たとえば 001 のように，受理される確率が 0 でも，1/2 以上でもないものが存在するからである．実際，001 が受理される確率は 1/16 である．　▨

例 11.15　計算時間が多項式時間であり，かつ，モンテカルロ性を有するランダム TM の例を紹介しよう．定義より，この TM は $\mathcal{RP}$ に属す言語を受理する．ここでは，形式的な記述の方法はとらない．入力は，グラフとして解釈される．問題は，グラフが内部に三角形を持つかどうか，すなわち，相異なる 2 頂点がすべて辺でつながれているような 3 頂点の集合が存在するかどうかを判定することである．三角形を持つグラフに対応する入力の全体が，考察の対象となる言語である．

モンテカルロ・アルゴリズムは，辺 (x, y) をランダムに選び，さらに，x, y 以外の頂点 z をランダムに選ぶという操作を繰り返す．辺や頂点の選択は，ランダム・テープ上のいくつかの新しいビットを見ることによって決定される．x, y, z が選択されるたびに，TM は，入力されたグラフが辺 (x, z) と辺 (y, z) を持つかどうかを調べる．もし，両方の辺を持つ場合は，TM は，入力されたグラフは三角形を持つ，と宣言する．

辺と頂点の選択を全部で k 回行う．TM は，そのうちの少なくとも 1 組について，それらが三角形をなすことがわかったら，入力を受理する．もし，どれも三角形にならなかったら，あきらめて，受理しないことにする．グラフが本当に三角形を持たない場合，k 個の選択のいずれについても，それが三角形を構成することはあり得ない．したがって，$\mathcal{RP}$ の定義の中の条件 (1) は満たさ

れる．すなわち，入力が，対象とする言語に属さない場合，受理の確率は 0 である．

グラフが n 個の頂点と e 個の辺を持つとする．グラフがある一つの三角形を持つとき，1 回の試行でその三角形の 3 頂点が選ばれる確率は $\left(\frac{3}{e}\right)\left(\frac{1}{n-2}\right)$ である．e 個の辺のうち 3 個の辺がこの三角形に含まれているし，それらの 3 個のうちの一つが選ばれた後，第 3 の頂点が選ばれる確率は $1/(n-2)$ だからである．この確率は小さい値であるが，我々はこの試行を k 回繰り返す．k 回の試行のうちの少なくとも 1 回の試行で三角形が生成される確率は，

$$1-\left(1-\frac{3}{e(n-2)}\right)^k \tag{11.4}$$

である．

よく使われる近似式で，小さい x に対し，$(1-x)^k$ は e^{-kx} にほぼ等しい，というものがある．ただし，$e = 2.718\cdots$ は自然対数の底である．ここで，たとえば，$kx = 1$ となる k をとれば，e^{-kx} は 1/2 より確実に小さい値となり，$1-e^{-kx}$ は 1/2 より確実に大きい値，もう少し正確に言うと，約 0.63 になる．したがって，$k = e(n-2)/3$ と定めれば，式 (11.4) より，三角形を持つグラフの受理の確率が 1/2 以上になることが保証される．よって，上記のアルゴリズムはモンテカルロ型である．

次に，この TM の計算時間を考えてみよう．e と n はどちらも入力の長さより大きいことはない．また，k は，その定め方より e と n の積の定数倍程度なので，入力の長さの 2 乗を超えることはない．各回の試行では，高々4 回入力を眺める（辺と頂点をランダムに選ぶために 2 回，さらに，2 個の辺の存在を調べるために 2 回）ので，その実行に要する時間は入力の長さに関する線型の値である．したがって，この TM は，入力の長さに関する 3 次式以下の計算時間ののちに停止する．すなわち，この TM は多項式の計算時間を持つこととなり，言語が $\mathcal{RP}$ に属すための 3 番目かつ最後の条件が満たされる．

以上より，三角形を持つグラフから成る言語はクラス $\mathcal{RP}$ に属すことが結論された．なお，この言語はクラス $\mathcal{P}$ にも属すことに注意されたい．三角形のすべての可能性を調べる体系的な探索を，多項式時間で行うことができるからである．11.4 節の冒頭で述べたように，$\mathcal{RP}-\mathcal{P}$ に属すように見える言語の例を

見つけることは，実はとても難しいことなのである． ▨

11.4.5 $\mathcal{RP}$ に属す言語の認識

言語 L を受理するための，多項式時間，モンテカルロ型チューリング機械 M があるとする．いま，入力 w が与えられ，w が L に属すかどうかを知りたいものとする．ランダム・ビットの生成を模倣するため，コイン投げか，または，別の乱数生成用の道具を用いて，w に対して M を動かしたとすると，我々は次のことを知る．

(1) w が L に属さないとき，TM の実行により w が受理に導かれることは絶対にない．

(2) w が L に属すとき，少なくとも 50％の確率で w は受理される．

この実行の結果を確定的なものとして単純に受け取ることにすると，本来，受理しなければならない w を受理しないという結果が生じる．（これを**偽否定** (false negative) の結果という．）しかし，本来，受理してはいけない w を受理してしまうことは決してない．（すなわち，**偽肯定** (false positive) の結果を得ることはない．）このように，我々は，ランダム TM 自身と，w が L に属すか否かを判定するために用いるアルゴリズムとの間を区別する必要がある．偽否定の結果を完全に除去することは不可能だが，同じテストを何度も繰り返し実行することにより，偽否定が生じる確率を，好きなだけ小さな値になるように，減少させることは可能である．

たとえば，偽否定の確率を 10 億分の 1 に減らしたければ，テストを 30 回繰り返して実行すればよい．w が L に属すとき，30 回のテストのすべてで w が受理されずに終わる確率は，2^{-30} を超えることはない．この値は，10^{-9}，すなわち，10 億分の 1 より小さい．一般に，偽否定の確率を $c > 0$ より小さい値にしたいときは，テストを $\log_2(1/c)$ 回実行すればよい．c が定数ならばこの回数も定数であり，また，L が $\mathcal{RP}$ に属すと仮定されていることから，ランダム TM M の 1 回の実行は多項式時間であるので，繰り返しのテストの全体に要する時間も多項式時間であることがわかる．以上の考察の結果を，定理として述べておく．

定理 11.16 L は $\mathcal{RP}$ に属すとする．このとき，任意の定数 $c > 0$ に対し，(c がどんなに小さな値であったとしても,) 与えられた入力 w が L に属すかどうかを判定する多項式時間のランダム・アルゴリズムで，偽肯定の誤りはまったくなく，偽否定の誤りの確率が c 以下であるものが存在する． ▨

11.4.6 クラス $\mathcal{ZPP}$

ランダム性に関わる言語のクラスで，我々が考察する第 2 のものは，**誤りゼロ・確率的・多項式** (zero-error, probabilistic, polynomial)，すなわち，$\mathcal{ZPP}$ と呼ばれるクラスである．このクラスは，どんな入力に対しても必ず停止し，停止するまでの平均実行時間が入力の長さに関する多項式の時間であるようなランダム TM に基づいて定められる．この TM は，受理状態に入る（そして，その時点で停止する）とき，入力を受理する．一方，受理状態に入らずに停止するとき，入力を受理しない．したがって，クラス $\mathcal{ZPP}$ の定義は，$\mathcal{P}$ の定義とほとんど同じである．異なるのは，$\mathcal{ZPP}$ では，TM の動作の中に乱数が含ま

$\mathcal{RP}$ の定義において分数 1/2 は特別の意味を持つのか?

我々は $\mathcal{RP}$ を定義する際，L に属す任意の文字列 w に対し w を受理する確率は 1/2 以上でなければならないという条件を課した．しかし，実は，1/2 の代わりに，0 より大きく 1 より小さい任意の定数を用いて $\mathcal{RP}$ を定義することもできたのである．定理 11.16 は，M による試行を適当な回数だけ繰り返すことにより，受理の確率を，1 より小さい範囲で，好きなだけ大きくすることができることを主張している．11.4.5 項で，L に属す文字列に対する非受理の確率を減少させるための手法を紹介したが，それと同じ手法により，L に属す w を受理する確率が 0 より大きな任意の確率であるランダム TM に対し，試行をある定数回繰り返すことによって，確率を 1/2 にまで押し上げることが可能である．

我々の $\mathcal{RP}$ の定義では，受理の確率として 1/2 を課するという条件は変えないが，クラス $\mathcal{RP}$ の定義の中で用いるためには，0 でない任意の確率で十分であることには注意する必要がある．ただし，1/2 以外の定数への変更を行うと，特定のランダム TM によって定義される言語は，一般に，変化してしまう．たとえば，仮に，要請される確率を 1/16 に低下させるとすると，例 11.14 で考察したランダム TM について，文字列 001 がランダム TM の言語に含まれることになってしまうことが容易に理解できる．

れることを許す点と，測定の対象が，最悪の場合の計算時間ではなく，平均実行時間である点の 2 点である．

常に正しい解を与えるが，計算時間はいくつかのランダム・ビットの値によって変化するような TM は，**ラスベガス** (Las-Vegas) **型テューリング機械**，または，ラスベガス・アルゴリズムと呼ばれることがある．この言葉を用いると，$\mathcal{ZPP}$ とは，多項式の平均計算時間を持つラスベガス型テューリング機械によって受理される言語の全体である，と考えることができる．

11.4.7 $\mathcal{RP}$ と $\mathcal{ZPP}$ の間の関係

これまでに定義してきたランダム性に関わる二つのクラスの間には，単純な関係が存在する．その関係を定理として述べるためには，まず最初に，クラスの補集合について考察することが必要となる．もし，L が $\mathcal{ZPP}$ に属すならば，$\overline{L}$ も $\mathcal{ZPP}$ に属すことは明らかであろう．その理由は，L が多項式の平均計算時間を持つラスベガス型 TM M によって受理されるとき，M に手を加えて，$\overline{L}$ を受理するように修正することが可能だからである．すなわち，M が受理して停止する場合は受理せずに停止するようにし，また，受理せずに停止する場合は受理状態に行って停止するように修正すれば，$\overline{L}$ は，その修正された TM によって受理される．

一方，$\mathcal{RP}$ が補集合をとる演算のもとで閉じているかどうかは自明ではない．なぜならば，モンテカルロ型テューリング機械の定義では，受理と非受理が対称に扱われてはいないからである．そこで，我々は，クラス補 $\mathcal{RP}$ を次のように定義することにする．**補** $\mathcal{RP}$(co-$\mathcal{RP}$) は，$\overline{L}$ が $\mathcal{RP}$ に属すような言語 L の全体からなる集合である．すなわち，補 $\mathcal{RP}$ は，$\mathcal{RP}$ に属す言語の補集合全体の集合である．

定理 11.17 $\mathcal{ZPP} = \mathcal{RP} \cap$ 補 $\mathcal{RP}$

証明 まず，$\mathcal{RP} \cap$ 補 $\mathcal{RP} \subseteq \mathcal{ZPP}$ を示す．L は $\mathcal{RP} \cap$ 補 $\mathcal{RP}$ に属すとする．すなわち，L と $\overline{L}$ は，ともに，計算時間が多項式のモンテカルロ型 TM を持つ．$p(n)$ を，両方の TM の計算時間を上から抑えるような十分大きな多項式であるとする．L に対するラスベガス型 TM M を次のように設計する．

(1) L に対するモンテカルロ型 TM を実行する．この TM が入力を受理する場合には，M も受理して停止する．

(2) そうでない場合は，$\overline{L}$ に対するモンテカルロ型 TM を実行する．この TM が入力を受理する場合には，M は受理せずに停止する．受理しない場合は，M は手順 (1) に戻る．

明らかに，M が入力 w を受理するのは，w が L に属す場合のみであり，また，M が w を受理しないのは，w が L に属さないときのみである．1 回分の実行（手順 1 と手順 2 の 1 回ずつの実行）に要する平均実行時間は $2p(n)$ 以下である．さらに，どの回についても，1 回分の実行で決着がつく確率は少なくとも 1/2 である．w が L に属すとき，手順 (1) により M による受理が導かれる確率は 50％以上であり，一方，w が L に属さないとき，手順 (2) により M による非受理が導かれる確率は 50％以上であるからである．したがって，M の平均実行時間は，

$$2p(n) + \frac{1}{2}2p(n) + \frac{1}{4}2p(n) + \frac{1}{8}2p(n) + \cdots = 4p(n)$$

を超えることはない．

さて，次に，いまの逆を考えよう．L は $\mathcal{ZPP}$ に属すと仮定し，L は $\mathcal{RP}$ と補 $\mathcal{RP}$ の両方に属すことを示す．仮定より，L は，平均計算時間がある多項式 $p(n)$ であるような，ラスベガス型 TM M_1 によって受理される．L に対するモンテカルロ型 TM M_2 を，次のように構成する．M_2 は M_1 の $2p(n)$ ステップ分の動作を模倣する．その間に M_1 が入力を受理する場合には，M_2 も受理する．そうでない場合は，M_2 は入力を受理しない．

長さ n の入力 w は L に属さないとする．このとき，M_1 が w を受理することは絶対にあり得ない．したがって，M_2 も w を受理することはない．次に，w は L に属すとしよう．このとき，M_1 はいつかは w を受理することになるであろうが，$2p(n)$ ステップ以内では受理することもあり得るし，受理しないこともあり得る．

しかし，我々は，M_1 が $2p(n)$ ステップ以内に w を受理する確率は 1/2 以上であることを示すことができる．いま，M_1 が $2p(n)$ ステップ以内に w を受理する確率は定数 $c < 1/2$ であると仮定する．このとき，M_1 の入力 w に対する平均計算時間は少なくとも $(1-c)2p(n)$ である．なぜならば，M_1 が $2p(n)$ より多くの時間を要する確率は $1-c$ であるからである．しかし，$c < 1/2$ という仮定より $2(1-c) > 1$ となり，M_1 の w に対する平均計算時間は $p(n)$ より大きいことになってしまう．M_1 の平均計算時間は高々$p(n)$ であるという仮定に対して矛盾が生じたので，したがって，M_2 が入力を受理する確率は少なくとも 1/2 であることが結論される．よって，M_2 は多項式時間有界モンテカルロ型 TM であることがわかり，L は $\mathcal{RP}$ に属すことが証明された．

L が補 RP にも属すことを証明するためには，いまと本質的に同じ構成を行い，最後に M_2 の出力を反転させておけばよい．すなわち，$\overline{L}$ を受理するには，M_1 が $2p(n)$

時間以内に受理しないことを決めたとき，M_2 は受理をし，それ以外のときは M_2 は受理をしない，と定めればよい．こうすれば，修正された M_2 は，$\overline{L}$ に対する多項式時間有界モンテカルロ型 TM となる．▨

11.4.8 クラス $\mathcal{P}$ と $\mathcal{NP}$ とに対する関係

定理 11.17 によれば，$\mathcal{ZPP} \subseteq \mathcal{RP}$ が成り立つ．これらの二つのクラスを，次の簡単な定理により，$\mathcal{P}$ と $\mathcal{NP}$ との間に位置づけることができる．

定理 11.18 $\mathcal{P} \subseteq \mathcal{ZPP}$

証明 任意の決定性多項式時間有界 TM は，ラスベガス型多項式時間有界 TM とみなすことができる．ランダムな選択を行う能力をたまたま使わないだけだと考えればよい．▨

定理 11.19 $\mathcal{RP} \subseteq \mathcal{NP}$

証明 L を受理する多項式時間有界モンテカルロ型 TM M_1 が与えられたとする．M_1 と同じ計算時間を持ち，L を受理する非決定性 TM M_2 を構成することができる．M_1 が新しいランダム・ビットを見るときには，M_2 は，非決定性で，そのビットに対する両方の可能な値のうちの一つを選択し，選択した値を，M_1 のランダム・テープを模倣するために用意された，自分自身のテープ上に書き留めておく．M_2 は，M_1 が受理するときには受理し，それ以外のときには受理しない．

w が L に属すとする．そのとき，M_1 が w を受理する確率は少なくとも 50％であるので，ランダム・テープ上のビット列の中に w を受理に導くものが必ず存在するはずである．M_2 は，いくつもある列の中から，そのビット列を選ぶことができ，したがって，w を受理することができる．すなわち，w は $L(M_2)$ に属す．一方，w が L に属さない場合には，ランダム・ビット列のどれも M_1 を受理に導くことはない．したがって，M_2 を受理に導く選択肢の列は存在しない．よって，w は $L(M_2)$ に属さない．▨

図 11.8 に，本節で導入したクラスとその他の「近隣」のクラスの間の関係を示す．

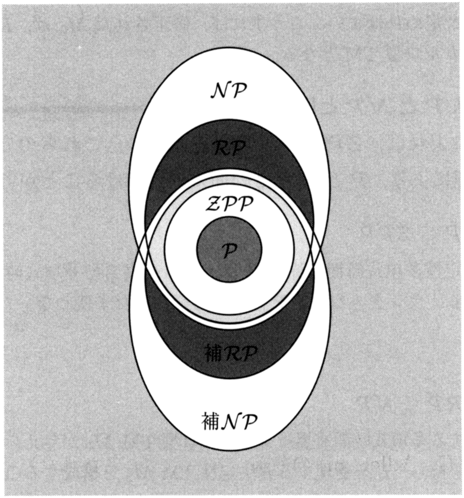


図 11.8 $\mathcal{ZPP}$ と $\mathcal{RP}$ の，他のクラスとの間の関係

11.5 素数性判定問題の計算量

本節では，次の特定の問題を取り上げる．与えられた整数が素数であるかどうかを判定せよ．まず，素数の集合や素数性判定問題を考察する動機づけを与えるため，コンピュータ・システムの安全性に対するそれらの本質的な役割に関する議論から始める．次に，素数全体の集合は，$\mathcal{NP}$ と補 $\mathcal{NP}$ の両方に属すことを示す．最後に，ランダム・アルゴリズムを考察し，素数全体の集合は $\mathcal{RP}$ にも属すことを示す[7]．

[7] ＜**訳注**：本書が出版された後，下記の論文において，素数全体の集合は $\mathcal{P}$ に属すことが証明された．M. Agrawal, N. Kayal and N. Saxena, “PRIMES is in $\mathcal{P}$”, 2002.＞

11.5.1　素数性判定問題の計算量

正の整数 p (> 1) が**素数** (prime) であるとは，p を割り切る正の整数は 1 と p だけしかないという意味である．正の整数 (> 1) が素数でないとき，それは**合成数** (composite) であるという．すべての合成数は，いくつかの素数の積として，順序を除いて，一意的に表される．

例 11.20　小さい順に例をあげると，2, 3, 5, 7, 11, 13, 17 などが素数である．整数 504 は合成数であり，その素因数分解は $2^3 \times 3^2 \times 7$ である．▨

コンピュータ・システムの安全性を高めるための技法がいくつか知られている．その中で，今日，もっともよく利用されている方法は，整数の素因数分解は困難である，という仮定に基礎を置くものである．整数の素因数分解とは，与えられた合成数に対しその素因子を求めよ，という問題である．とくに，RSA コード（この技法の創始者である R. リベスト (R. Rivest)，A. シャミール (A. Shamir)，L. エイデルマン (L. Adleman) の名前に由来して命名された）と呼ばれるものに基礎を置く方法では，たとえば，各々がおよそ 64 ビットの長さの二つの素数の積である 128 ビットの整数が用いられる．以下，素数が重要な役割を演じるシナリオを二つ紹介しよう．

公開鍵暗号系　オンライン書店から本を購入したいものとする．売り手は，買い手のクレジット・カードの番号を尋ねてくる．しかし，番号を一定の書式にタイプして，その書式を電話線やインターネットを通して売り手に伝えるという方法をとるのは，あまりにも危険である．第三者が電話線をひそかに盗聴しているかも知れないし，あるいは，インターネット上を運ばれていくパケットが途中で横取りされてしまうかも知れないからである．クレジット・カードの番号を第三者にのぞき見られるのを防ぐため，売り手は買い手のブラウザに鍵 k を送る．この鍵は二つの素数の積として得られる整数であり，売り手のコンピュータがこの目的だけのために生成したものである．大きさは 128 ビットであるかもしれない．買い手のブラウザは，鍵 k と買い手が暗号化する必要のあるデータ x とに対し，関数 $y = f_k(x)$ を適用する．RSA システムの一部である関数 f は，盗聴する可能性のある人も含めて，一般に知られてしまってい

るかもしれない．しかし，k の因数分解を知らずには，$x = f_k^{-1}(y)$ を満たす逆関数 f_k^{-1} を k の長さに関する指数関数より少ない時間で計算することはできない，と信じられている．

したがって，盗聴者が，たとえ，y を知ってしまい，かつ，f がどのような関数であるかを知ってしまったとしても，k がいくつであるかを知って，さらに，k の因数分解を行うことができない限りは，盗聴者は，いまの場合クレジット・カードの番号である x を復元することはできない．一方，売り手は，まず二つの素数を求めることから始めたので，当然，鍵 k の因数分解を知っている．その知識を使うと，容易に f_k^{-1} を適用することができ，y から x を復元することができる．

公開鍵による署名　RSA コードが開発された際の，本来のシナリオは次のようなものであった．電子メールの送受信の際，受け取り人が，その電子メールは確かに文面に記されている送り手からのものであるということを容易に確認できるようにするため，そして，第三者が送り手の名前を「偽造」することができないようにするため，電子メールに「署名」をしたい．たとえば，ある人が，$x =$“私はサリー・リーさんに 10 ドル支払うことを約束する” というメッセージに署名をしたいとする．しかし，サリーさん自身が自分でそのような署名つきのメッセージを作成することができては困るし，また，当事者が知らない間に，第三者にそのような署名つきのメッセージを作成されてしまっても困る．

これらの目的を達するため，各人が，素因数分解はその人だけが知っているような鍵 k を選ぶ．鍵 k は，例えば，各人のウェブ・サイト等で，公開される．したがって，関数 f_k を任意のメッセージに適用することはだれでもできる．さて，ある人が，メッセージ x に署名をして，それをサリーさんに送りたいとする．その送り手は，まず，$y = f_k^{-1}(x)$ を計算して，x の代わりに y をサリーさんに送る．サリーさんは，送り手のウェブ・サイトから**公開鍵** (public key) である f_k を見つけて，それを用いて $x = f_k(y)$ を計算する．このようにして，サリーさんは，送り手が本当に 10 ドルを支払うことを約束したことを知ることができる．

もし，送り手がメッセージ y を送ったことを否定した場合には，サリーさんは裁判官の前で，その送り手だけが関数 f_k^{-1} を知っていること，そして，サリーさんや別の第三者がその関数を発見することは「不可能」であることを主張することができる．真の送り手だけが y を作り出すことができたのである．このシステムは，二つの大きな素数の積である整数に対し，その整数を因数分解することは難しすぎて事実上不可能である，という，確からしいけれどもまだ証明されてはいない仮定に基礎を置くシステムである．

素数性判定問題の計算量に関する要請　上の二つのシナリオは，どちらもうまく機能し，二つの大きな素数の積を因数分解するには，実際，指数時間を要するという意味で，安全であると信じられている．我々が本章や第 10 章で学んだ計算量理論は，安全性や暗号論の学習に対し，次の二つの意味で関わりを持つ．

(1)　公開鍵の構成のためには，大きな素数を短時間で見つけ出すことが要求される．整数論における基礎的な結果の一つに，n ビットの整数に対しそれが素数である確率は $1/n$ のオーダである，という事実がある．したがって，もし，n ビットの整数が素数であるかどうかを判定する多項式時間（素数自身の値に関してではなく，n に関する多項式）のアルゴリズムがあるとすれば，我々は，ランダムに整数を選び，それが素数であるかどうかを判定する作業を繰り返し，素数が見つかったら停止する，という手順を踏めばよい．それにより，素数を見つけるための多項式時間ラスベガス型アルゴリズムが得られる．n ビットの素数を見い出すまでに要する判定作業の平均実行回数は，およそ n だからである．たとえば，64 ビットの素数を求めたい場合，平均して約 64 個の整数についてテストを行うことになる．もちろん，運の悪いときは，それよりはるかにたくさんの回数のテストが必要になることもあり得る．現時点では，残念ながら，素数性を判定するテストで多項式時間であることが保証されたものはない[8]．ただし，11.5.4 項で見るように，モンテカルロ型のアルゴリズムで多項式時間のものは存在する．

---

[8] <**訳注**：前に記したように，素数全体の集合は $\mathcal{P}$ に属すことが証明されている．>

(2) RSA 系統の暗号系の安全性は，一般に，因数分解を行うアルゴリズムで,（鍵のビット数に関して）多項式時間のものは存在しそうにないこと，とりわけ，ちょうど二つの大きな素数の積であることがわかっている整数を多項式時間で因数分解する方法は存在しそうにないことに基づく．仮に，素数全体の集合は NP 完全であること，さらには，合成数全体の集合は NP 完全であることを示すことができれば，我々はとても幸せな状況となる．なぜならば，そうなれば，多項式時間の因数分解アルゴリズムの存在は，これらの両方の言語に対する多項式時間のテストを与えることとなり，したがって，成り立ちそうにない $\mathcal{P} = \mathcal{NP}$ が証明されることになるからである．11.5.5 項で見るように，素数全体の集合と合成数全体の集合はどちらも $\mathcal{NP}$ に属す．それらは互いに他の補集合になっているので，仮に，いずれか一方が NP 完全であることがわかれば，そのことから $\mathcal{NP}$ = 補 $\mathcal{NP}$ が導かれてしまう．しかし，それも成り立ちそうにないことである．さらにまた，素数全体の集合が $\mathcal{RP}$ に属すという事実は，もし，素数全体の集合が NP 完全であることが示されたならば，そのことから，$\mathcal{RP} = \mathcal{NP}$ が結論される，ということを意味している．これもまた，上と同様に，成り立ちそうにない事柄である．

11.5.2 モジュラ算術の基礎

素数の集合を認識するアルゴリズムの考察に進む前に，**モジュラ算術** (modular arithmetic) に関する基礎的な事柄をいくつか紹介しよう．モジュラ算術とは，通常の算術演算をある整数，たいていは素数，を法として行うものである．p を任意の整数とする．p を法とする整数とは，$0, 1, \ldots, p-1$ である．

p を法とする加算と乗算は，上の p 個の整数の集合だけに適用されるが，それは次のように定義される．通常の加算または乗算の計算を行い，得られた結果を p で割り，余りを取り出す．加算は，極めて単純である．2 数の和は，p より小さいか，または，p 以上で $2p-2$ 以下であるかのどちらかである．前者の場合はそれ以上何もする必要はなく，後者の場合は，p を引いて，$0, 1, \ldots, p-2$ の範囲の整数を得ればよい．モジュラ加算は，通常の代数的法則にしたがう．すなわち，可換法則，結合法則を満たし，0 を単位元とする．減算は，加算の逆演算である．モジュラ差 $x - y$ を計算するには，まず，通常の減算を行い，その

結果が負数になったときは p を加えればよい．x の負数，すなわち，$-x$ は，通常の算術のように，$0-x$ と同じである．したがって，$-0=0$ であり，$x \neq 0$ のときは $-x$ は $p-x$ と同じである．

例 11.21　$p=13$ とする．このとき，$3+5=8$ であり，また，$7+10=4$ である．後者は次のようにして計算される．まず，通常の計算により，$7+10=17$ を求める．17 は 13 より小さくはない．したがって，13 を引き，4 を得る．次に，13 を法とする -5 の値は，$13-5$，すなわち，8 である．13 を法とする差 $11-4$ は 7 であり，一方，差 $4-11$ は 6 である．後者は，まず，通常の算術により $4-11=-7$ を求め，負数が得られたので 13 を加えて 6 を得ればよい．▨

p を法とする乗算は，次のように計算すればよい．まず，通常の整数のように乗算を行う．次に，p で割って，余りをとる．モジュラ乗算も，やはり，通常の代数的法則を満たす．可換法則，結合法則が成り立ち，1 が単位元であり，0 が零元である．さらに，加算との間で分配法則が成り立つ．ただし，零でない要素による除算には，注意が必要である．p を法とする整数に対しては，逆数の存在性すら，p が素数であるかどうかに依存する．一般に，x が p を法とする整数であるとき，すなわち，$0 \leq x < p$ であるとき，x^{-1}，あるいは，$1/x$ は，法 p で $xy=1$ を満たす y が存在するならば，その y である．そのような y が存在しないとき，x^{-1} は存在しない．

例 11.22　図 11.9 に，素数 7 を法とする，零でない整数に対する乗算表を示す．第 i 行，第 j 列の要素は，7 を法とする積 ij である．零でない整数はすべて逆数を持つことに注意されたい．2 と 4 は互いに他の逆数であり，3 と 5 も同様であるが，1 と 6 はどちらも自分自身が逆数である．すなわち，2×4, 3×5, 1×1 そして 6×6 はすべて 1 である．したがって，零でない任意の y で割ることができる．x/y を計算するには，まず，y^{-1} を求め，次に，$x\times y^{-1}$ の乗算を実行すればよい．たとえば，$3/4=3\times 4^{-1}=3\times 2=6$ である．

比較のため，6 を法とする乗算表を見てみよう．まず目に付くのは，1 と 5 だけしか逆数を持つものがないことである．1 と 5 はいずれも自分自身の逆数である．その他の数は逆数を持たない．さらに付け加えると，0 でない 2 数で，そ

1	2	3	4	5	6
2	4	6	1	3	5
3	6	2	5	1	4
4	1	5	2	6	3
5	3	1	6	4	2
6	5	4	3	2	1

図 11.9 7 を法とする乗算

1	2	3	4	5
2	4	0	2	4
3	0	3	0	3
4	2	0	4	2
5	4	3	2	1

図 11.10 6 を法とする乗算

の積が 0 になるものがある．たとえば，2 と 3 がそうである．このような現象は，通常の整数の算術では決して起こらないことであるし，また，素数を法とする算術においても決して起こらないことである． ▨

素数を法とする乗算と合成数を法とする乗算の間には，もう一つの顕著な違いがある．それは，素数性判定のテストにとって極めて重要なものである．p を法とする整数 a の**次数** (degree) とは，1 に等しくなる a の正のベキ数で最小のものである．ここでは証明をしないが，いくつかの重要な性質を列挙する．

- 素数 p と整数 a に対し，法 p で $a^{p-1} = 1$ が成り立つ．この命題は**フェルマーの定理** (Fermat's theorem) と呼ばれる[9]．
- 素数 p を法とする整数 a の次数は，常に，$p-1$ の約数である．
- 素数 p に対し，常に，p を法とする次数が $p-1$ であるような整数 a が存在する．

例 11.23 図 11.9 に示す，7 を法とする乗算表をもう一度見てみよう．2 の次数は 3 である．$2^2 = 4$ かつ $2^3 = 1$ だからである．3 の次数は 6 である．$3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$ かつ $3^6 = 1$ だからである．同様の計算により，4 の次数は 3，5 の次数は 6，6 の次数は 2，そして，1 の次数は 1 であることがわかる． ▨

---

9) ここでいうフェルマーの定理を,「フェルマーの最終定理」と混同しないよう注意されたい．フェルマーの最終定理とは，$n \geq 3$ に対し，$x^n + y^n = z^n$ が（自明でない）整数解を持たないことを主張する定理である．

11.5.3 モジュラ算術計算の計算量

モジュラ算術の素数性判定テストへの応用に進む前に，基本的な演算の計算時間について，基礎的な事柄を調べておこう．ある素数 p を法とする計算について考える．p の 2 進表現は n ビットの長さであるとする．すなわち，p 自身の大きさは，およそ 2^n である．いつものように，計算のための所要時間は，入力の長さ n に関して記述される．入力の「値」である p に関して記述されるのではない．たとえば，1 から p まで一つずつ数え上げていく作業には $O(2^n)$ の時間が必要となる．このように，p ステップを要する任意の計算は，n の関数として，多項式時間の計算ではない．

しかし，二つの数の p を法とする加算は，通常のコンピュータ上でも，あるいは，多テープ TM 上でも，$O(n)$ 時間で確かに実行することができる．単に，二つの 2 進数の加算を行い，その結果得られる値が p 以上であるときは，そこから p を引けばよい．同様に，二つの数の p を法とする乗算は，コンピュータ上でも，あるいは，テューリング機械上でも，$O(n^2)$ 時間で実行することができる．通常の方法で整数に対する乗算を行い，高々 $2n$ ビットの結果を得たのち，その結果を p で割り，余りをとればよい．

一方，整数 x のベキ乗を求める計算には，指数自身が n に関する指数関数であるかもしれないので，少々の工夫が必要となる．後に見るように，我々にとって，x の $p-1$ 乗を求める計算は大変重要な計算となる．$p-1$ はおよそ 2^n の大きさなので，x に x を $p-2$ 回掛けるようなことをしてしまうと $O(2^n)$ 回の乗算が必要となり，たとえ，1 回ごとの乗算は n ビットの整数に対する通常の乗算なので $O(n^2)$ の時間で実行できるとしても，全体としての計算時間は $O(n^2 2^n)$ になってしまう．これは，n に関する多項式ではない．

幸いなことに，「再帰的 2 乗法」とでも称すべき方法があり，x^{p-1}（または，x の，p 以下の任意のベキ乗）を n に関する多項式の時間で計算することができる．それは，次のような方法である．

(1) x のベキ乗 $x, x^2, x^4, x^8, \ldots$ を，指数が $p-1$ を超える直前まで，高々 n 個計算する．各々の値は，列の中の一つ前の値を 2 乗することによって得られる n ビットの整数であり，個々の計算時間は $O(n^2)$ である．したがって，計算時間の全体は $O(n^3)$ である．

(2) $p-1$ の 2 進表現，$p-1=a_{n-1}\cdots a_1a_0$，を求める．$p-1$ は次のように表される．

$$p-1=a_0+2a_1+4a_2+\cdots+2^{n-1}a_{n-1}$$

ここで，各 a_j は 0 または 1 である．したがって，

$$x^{p-1}=x^{a_0+2a_1+4a_2+\cdots+2^{n-1}a_{n-1}}$$

であり，この値は，$a_j=1$ であるような j に対する x^{2^j} の値全体の積である．手順 (1) において，すべての x^{2^j} は求められている．それらは n ビットの整数であるので，それらの中の n 個またはそれ以下の個数のものの積は，$O(n^3)$ 時間で計算することができる．

以上より，x^{p-1} の計算の全体に要する時間は $O(n^3)$ 時間であることがわかる．

11.5.4 ランダム多項式時間の素数性判定テスト

さて，準備ができたので，ランダム計算を利用して，大きな素数を求める方法について考えてみよう．もう少し詳しく言うと，ここで示したいのは，合成数全体から成る言語は $\mathcal{RP}$ に属す，という事実である．n ビットの素数を生成するために実際に用いられる方法は，n ビットの整数をランダムに取り出し，その整数に対し合成数を認識するモンテカルロ・アルゴリズムを多数回，たとえば，50 回，適用するという方法である．もし，そのうちのどれかのテストで，対象となる整数は合成数である，という結果が得られれば，その整数は素数ではないことがわかる．また，もし，50 回のテストのすべてにおいて，対象となる整数は合成数であるという結果が得られなければ，その整数が合成数である確率は 2^{-50} より小さい値である．すなわち，その場合，その整数は素数であると断定して，安全性に関する演算をそのことを基にして行うことにしても，ほぼ間違いはないと考えられる．

ここでは，アルゴリズムを完全な形で詳述することはしない．その代わり，ごくわずかな例外を除いて機能する，アルゴリズムの本質的な部分のアイデアを説明することにする．フェルマーの定理を復習しよう．それは，p が素数であるならば，p を法として $x^{p-1}=1$ が常に成り立つ，というものであった．ま

た，さらに，次の事実も知られている．p が合成数であるとき，p を法として $x^{p-1} \neq 1$ となる x が一つでも存在すれば，1 以上 $p-1$ 以下の範囲に属す整数 x のうち，少なくとも半分の個数の x に対し $x^{p-1} \neq 1$ が成り立つ．

合成数に対するモンテカルロ・アルゴリズムとして，我々が用いるのは，次のようなものである．

(1)　1 以上 $p-1$ 以下の範囲に属す整数 x をランダムに選ぶ．

(2)　p を法として x^{p-1} を計算する．11.5.3 項の最後に述べたように，p を n ビットの整数とするとき，この計算は $O(n^3)$ 時間で行うことができる．

(3)　p を法として $x^{p-1} \neq 1$ であるならば，x を受理する．すなわち，x は合成数である．そうでない場合，受理せずに停止する．

p が素数であれば，常に，$x^{p-1} = 1$ が成り立ち，したがって，受理せずに停止する．これは，モンテカルロ型のアルゴリズムに対して要請される条件の一つ，すなわち，入力が目的の言語に属さないとき，その入力は決して受理されない，という条件が満たされていることを示す．一方，ほとんどすべての合成数に対し，少なくとも半分の個数の x は $x^{p-1} \neq 1$ を満たす．したがって，このアルゴリズムの 1 回の適用について，少なくとも 50％の受理の確率がある．これは，アルゴリズムがモンテカルロ型であるために満たすべき条件のうちのもう一つのものに該当する．

ここまで述べてきたことにより，法 c で $x^{c-1} = 1$ を満たす少数の合成数 c の存在を除外すると，1 以上 $c-1$ 以下の範囲に属す大多数の x，とくに，c と共通の素因子を持たないすべての x について，合成数の集合は $\mathcal{RP}$ に属すことを実証したことになる．上記の例外的な合成数は**カーマイケル数** (Carmichael number) と呼ばれるが，これらの数に対してそれらが合成数であることを確かめるためには，(ここではその説明を省略するが,) 上とは別のもっと複雑なテストが必要となる．最小のカーマイケル数は 561 である．我々は，3, 11, 17 のいずれでも割り切れないすべての x に対し，法 561 で $x^{560} = 1$ が成り立つことを示すことができる．それにも関わらず，$561 = 3 \times 11 \times 17$ は明らかに合成数である．以上より，証明は一部が不完全な形でしか述べてないが，次の定理が成り立つ．

因数分解をランダム多項式時間で行うことは可能か？

11.5.4 項に示したアルゴリズムは，与えられた整数が合成数であることは教えてくれるかもしれないが，合成数を因数分解する方法までは教えてくれないことに注意されたい．多項式時間で因数分解を行う方法は存在しない，と一般に信じられている．たとえ，ランダム性の利用を許しても，あるいは，計算時間に関する制約を緩めて平均計算時間が多項式時間という制約に変えたとしても同様である．仮に，この仮定が正しくないとすると，11.5.1 項に述べた応用は安全性を欠くものとなり，現実の使用には堪えないものとなる．

定理 11.24　合成数全体の集合は $\mathcal{RP}$ に属す．　▨

11.5.5　非決定性の素数性判定テスト

素数性の判定に関する結果で，興味深く，また，重要なものをもう一つ取り上げよう．それは，素数全体の集合は $\mathcal{NP} \cap$ 補 $\mathcal{NP}$ に属す，という性質である．なお，合成数全体の集合は素数全体の集合の補集合であることから，上の性質より，合成数全体の集合も $\mathcal{NP} \cap$ 補 $\mathcal{NP}$ に属すことが導かれる．この事実の重要性は，素数全体の集合も，また，合成数全体の集合も NP 完全ではありそうもないことが示唆されるという点にある．なぜならば，どちらかが NP 完全であれば，成り立たないと予想されている等式 $\mathcal{NP} =$ 補 $\mathcal{NP}$ が成り立つことになってしまうからである．

証明したいことの一方は簡単である．合成数全体の集合は，容易にわかるように，$\mathcal{NP}$ に属し，したがって，素数全体の集合は補 $\mathcal{NP}$ に属す．そのことを，まず，証明しよう．

定理 11.25　合成数全体の集合は $\mathcal{NP}$ に属す．

証明　合成数を受理する，非決定性で多項式時間のアルゴリズムは，次のようなものである．

(1)　n ビットの整数 p が与えられたとき，高々 n ビットの約数 f を推測する．ただし，$f = 1$，または，$f = p$ を選んではいけない．この部分は非決定性で実行

され，一連の選択の系列に沿って，可能な f のすべてが推測される約数の候補となる．計算時間は，どの選択の系列に対しても $O(n)$ 時間である．

(2) p を f で割り，余りが 0 であるかどうかを確かめる．余りが 0 ならば，受理する．この部分は決定性で実行される．計算時間は，多テープ TM 上で $O(n^2)$ 時間である．

p が合成数ならば，p は，1 と p 以外の約数 f を少なくとも一つ持つはずである．NTM は，n ビット以下のすべての可能な整数を推測するので，ある選択の系列で f を推測するはずである．この系列は，受理に至る系列である．逆に，NTM によって受理されるということは，1 と p 以外の p の約数が見つかったということを意味する．

以上により，上記の NTM は，合成数の全体，そして，それだけから成る言語を受理することがわかった． ▨

素数全体の集合を認識する NTM の構成は，いまよりももっと難しい．整数が素数ではない理由（約数）を推測し，その推測が正しいことを確認することは可能であったが，さて，それでは，整数が素数で**ある**理由を「推測」するにはどうすればよいのだろうか．非決定性で多項式時間のアルゴリズムは，次の事実に基づいて構成される．(この事実について，ここでは紹介をするだけで，証明はしない.) p が素数であるとき，1 と $p-1$ の間の整数 x で，法 p に関する次数が $p-1$ のものが存在する．たとえば，例 11.23 で見たように，素数 $p=7$ に対し，3 と 5 の次数はいずれも 6 である．

NTM の非決定性の機能を用いれば，整数 x は容易に推測することができる．しかし，x の次数が $p-1$ であることを確認する方法は自明という訳ではない．その理由は，単純に「次数」の定義をそのまま適用しようとすると，$x^2, x^3, \ldots, x^{p-1}$ のすべてについて，どれも 1 に等しくはないということを確かめなければならなくなるからである．その作業を行うには，$p-3$ 回の乗算を実行することが必要となり，したがって，p を n ビットの整数とするとき，少なくとも 2^n の時間が必要となる．

もっとよい方法は，さらにもう一つの事実を利用することである．(この事実についても，ここでは紹介をするだけで，証明は記さない.) 素数 p を法とする x の次数は $p-1$ の約数である．したがって，$p-1$ の素因子をすべて知ること

ができれば[10]，$p-1$ の素因子 q の各々について，$x^{(p-1)/q} \neq 1$ を確認するだけで十分となる．これらの x のベキ乗がすべて 1 に等しくないならば，x の次数は $p-1$ 以外ではあり得ない．これらのテストの回数は $O(n)$ であり，したがって，これらのすべてを多項式時間のアルゴリズムで実行することができる．もちろん，$p-1$ の素因数分解を容易に実行することはできない．しかし，非決定性で，$p-1$ の素因子を**推測**することは可能である．すべての素因子を推測した後，次の処理を行えばよい．

a)　それらの積は，実際，$p-1$ であることを確認する．

b)　各々が素数であることを確認する．そのためには，いま設計しようとしている非決定性，多項式時間のアルゴリズムを再帰的に用いればよい．

アルゴリズムの詳細，および，それが非決定性，多項式時間のアルゴリズムであることの証明は，下記の定理の証明の中に記す．

定理 11.26　素数全体の集合は $\mathcal{NP}$ に属す．

証明　n ビットの整数 p が与えられたとき，次の処理を行う．最初に，n が 2 以下（すなわち，p が 1, 2, 3 のいずれか）の場合は，直接，質問に答える．すなわち，2 と 3 は素数であり，1 は素数ではない．それ以外の場合は，次の手順にしたがう．

(1)　素因子のリスト $(q_1, q_2, \ldots, q_k)$ を推測する．それらの 2 進表示の長さの和は高々$2n$ ビットであり，各々の長さは $n-1$ ビット以下である．リストの中に同じ素数が重複して出現することを許す．$p-1$ の因子として，ある素数が 1 より大きなベキ乗の形で現れているかもしれないからである．たとえば，$p=13$ のとき，$p-1=12$ の素因子のリストは (2, 2, 3) となる．この部分の処理は，非決定性で行われる．各々の枝（選択の列）の計算時間は $O(n)$ である．

(2)　リストの中の q_j をすべて掛けあわせ，全体の積が $p-1$ であることを確かめる．この部分は，決定性で行われ，計算時間は $O(n^2)$ を超えることはない．

(3)　全体の積が $p-1$ であるとき，ここに記述しているアルゴリズムを再帰的に用いて，各々が素数であることを確かめる．

(4)　q_j がすべて素数であるならば，x の値を推測し，すべての q_j に対して

---

[10] p が素数であるとき，$p=3$ のつまらない場合を除き，$p-1$ は決して素数にはならない．2 以外のすべての素数は奇数だからである．

$x^{(p-1)/q_j} \neq 1$ であることを確かめる．このテストにより，x の p を法とする次数は $p-1$ であることが保証される．なぜならば，もしそうでないとすれば，その次数は少なくとも一つの $(p-1)/q$ の約数でなければならないが，そうなっていないことが上で確かめられたからである．ここで，x を，次数の任意の倍数乗した値は，常に 1 であることに注意されたい．ベキ乗の計算は，11.5.3 項で述べた方法により，効率よく行うことができる．高々 k 回のベキ乗の計算が行われるが，明らかに $k \leq n$ であり，また，1 回のベキ乗の計算は $O(n^3)$ 時間で行うことができるので，このステップの実行に要する計算時間は全体で $O(n^4)$ 時間である．

最後に，この非決定性のアルゴリズムは，多項式時間のアルゴリズムであることを示す必要がある．再帰的ステップの (3) を除く各ステップは，任意の非決定性の枝に沿って進むとき，高々$O(n^4)$ の計算時間で十分である．再帰的な処理の部分は込み入っているので，再帰呼び出しの様子を，図 11.11 に示す木のように，視覚化して考えてみる．根には，我々の考察の対象である n ビットの素数 p を置く．根の下にある子の頂点には，$p-1$ の素因子として推測された q_j を並べる．これらについても，やはり，素数であることを証明する必要がある．各 q_j の下には，q_j-1 の素因子として推測された数を並べる．これらも，素数であることを証明する必要のある数である．以下同様にして，高々 2 ビットの整数に到達するまで，同様の作業を繰り返す．高々 2 ビットの整数が，葉の頂点となる．

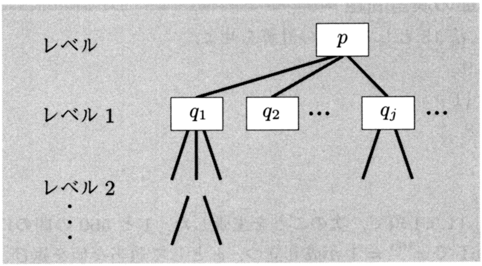


図 11.11　定理 11.26 のアルゴリズムによって実行される再帰呼び出しの列により，高さと幅がともに n 以下の木が形成される．

任意の頂点について，その子の頂点全体の積は，その頂点自身の値より小さい．このことから，根からの任意の深さ（レベル）について，その深さにあるすべての頂点の値の積は p 以下であることがわかる．値 i を持つ頂点で行われる仕事の計算時間は，再帰呼び出しで行われる仕事を除くと，ある定数 a に対し，高々$a(\log_2 i)^4$ である．その理由は，この仕事に要する計算時間は，すでに見たように，対象となっている値を2進数として表現するために必要なビット数の4乗のオーダとなるからである．

したがって，任意の1つのレベルについて，そのレベルで必要な仕事の計算時間に関する上界を求めるには，和 $\sum_j a(\log_2(i_j))^4$ の最大値を，積 $i_1 i_2 \cdots$ は p 以下であるという条件のもとで評価すればよい．4乗の式は凸であるから，最大値が得られるのは，すべての値が一つの i_j に含まれているときである．$i_1 = p$ であり，他に i_j がないとき，和は $a(\log_2 p)^4$ である．n は p の2進表示におけるビット数であり，したがって，$\log_2 p$ は高々 n であるので，$a(\log_2 p)^4$ は高々 an^4 である．

以上より，我々は，各々の深さ（レベル）で必要となる仕事の計算時間は，高々$O(n^4)$である，という結論に達する．レベルの個数は n 以下であるので，p が素数であるかどうかを調べる非決定性のテストのどの枝（選択の列）についても，$O(n^5)$ の計算時間で十分である．▨

以上より，素数全体の集合とその補集合の両方が $\mathcal{NP}$ に属すことがわかった．もし，仮に，それらのいずれかが NP 完全であるとすると，定理11.2により，我々は，$\mathcal{NP} =$ 補 $\mathcal{NP}$ の証明を得ることとなる．

11.5.6　11.5節の演習問題

♣ **問 11.5.1**　法13として，次の計算をせよ．

a)　11+9

*b)　$9-11$

c)　5×8

*d)　5/8

e)　5^8

♣ **問 11.5.2**　11.5.4項で，次のことを主張した．1と560の間のほとんどの x に対し，法561で $x^{560} = 1$ が成り立つ．x として適当な値を選び，この等式が成り立つことを確かめよ．なお，計算をする際には，乗算を559回行うことを避けるため，11.5.3項で述べたように，まず，560の2進数表示を求め，次に，いろいろな j の値に対し，法561に関する x^{2^j} を計算するようにせよ．

♣ **問 11.5.3** 1 と $p-1$ の間の整数 x に対し，1 と $p-1$ の間の整数 y で $y^2=x$ を満たすものが存在するとき，x は p を法とする**平方剰余** (quadratic residue) であるという．

*a) 7 を法とする平方剰余をすべて求めよ．解答の参考にするため，図 11.9 の表を用いてもよい．

b) 13 を法とする平方剰余をすべて求めよ．

!c) p を素数とするとき，p を法とする平方剰余の個数は $(p-1)/2$ であることを示せ．すなわち，p を法とする非負整数のうち，ちょうど半分のものが平方剰余である．**ヒント**：設問 (a)，(b) に対する読者の解答を検討してみよ．どの平方剰余も二つの相異なる整数の平方となっていることについて，その理由を説明するパターンを見い出せ．p が素数であるとき，一つの整数が三つの相異なる整数の平方となることはあり得るか．

11.6 第11章の要約

◆**クラス 補 $\mathcal{NP}$**：言語は，その補集合が $\mathcal{NP}$ に属すとき，補 $\mathcal{NP}$ に属すといわれる．$\mathcal{P}$ に属すすべての言語は，確かに，補 $\mathcal{NP}$ に属す．しかし，$\mathcal{NP}$ に属す言語で補 $\mathcal{NP}$ には属さないものや，逆に，補 $\mathcal{NP}$ に属す言語で $\mathcal{NP}$ には属さないものが存在すると思われる．とくに，NP 完全問題は補 $\mathcal{NP}$ に属さないと思われる．

◆**クラス $\mathcal{PS}$**：言語は，次のような性質を持つ非決定性 TM によって受理されるとき，$\mathcal{PS}$（多項式領域）に属すといわれる．ある多項式 $p(n)$ が存在して，長さ n の任意の入力に対し，TM はテープ上のます目を $p(n)$ 個を超えて使用することはない．

◆**クラス $\mathcal{NPS}$**：$\mathcal{PS}$ と同様に，テープの使用量が入力の長さに関する多項式によって限定されているような，非決定性 TM による受理を定義することもできる．このような方式で受理される言語のクラスを $\mathcal{NPS}$ と称する．ただし，サビッチの定理により，$\mathcal{PS}=\mathcal{NPS}$ が成り立つことがわかる．詳しくいうと，領域量が $p(n)$ の NTM を，領域量が $p^2(n)$ の DTM によって模倣することができる．

◆**ランダム・アルゴリズムとテューリング機械**：多くのアルゴリズムで，ランダム性が有効に利用される．実際のコンピュータでは，「コイン投げ」を模倣するために乱数生成ルーチンが用いられる．ランダム・テューリング機械は，ランダム・ビットの列を書き込むためのテープを追加して付与されると，同様のランダムな振る舞いを行うことができる．

◆**クラス $\mathcal{RP}$**：言語がランダム・多項式時間で受理されるとは，多項式時間のランダム・テューリング機械で，入力がその言語に属すときは，その入力を 50％以上の確

率で受理するものが存在することである．入力がその言語に属さないときは，この TM は決して入力を受理しない．このような TM，あるいは，アルゴリズムは,「モンテカルロ」型と呼ばれる．

- ◆**クラス $\mathcal{ZPP}$**：言語が，誤りゼロ・確率的・多項式時間のクラスに属すのは，言語の所属に関して常に正しい判定を与えるランダム・テューリング機械によって受理されるときである．この TM の計算時間は，最悪の場合は多項式を超えるかもしれないが，平均計算時間は多項式時間である．このような TM，あるいは，アルゴリズムは,「ラスベガス」型と呼ばれる．
- ◆**言語のクラスの間の関係**：補 $\mathcal{RP}$ というクラスは，$\mathcal{RP}$ に属す言語の補集合から成る集合である．次のような包含関係が知られている．$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq (\mathcal{RP} \cap$ 補 $\mathcal{RP})$．また，$\mathcal{RP} \subseteq \mathcal{NP}$ が成り立ち，したがって，補 $\mathcal{RP} \subseteq$ 補 $\mathcal{NP}$ も成り立つ．
- ◆**素数全体の集合と $\mathcal{NP}$**： 素数全体の集合も，また，その補集合 — 合成数全体の集合 — も $\mathcal{NP}$ に属す．これらの事実から，素数全体の集合や合成数全体の集合は NP 完全ではなさそうであることが示唆される．素数を基礎とする重要な暗号系があるが，上の集合の NP 完全性の証明がもし存在すれば，それは，安全性に関する強い証拠を与えるものとなる．
- ◆**素数全体の集合と $\mathcal{RP}$**：合成数全体の集合は $\mathcal{RP}$ に属す．合成数であることをテストするランダム・多項式のアルゴリズムは，大きな素数の生成，もう少し正確に言うと，合成数である確率が任意に小さな値であるような大きな整数の生成，を目的として，広く利用されている．

11.7 第 11 章の参考文献

文献 [2] により，テューリング機械が使用する領域量を限定することによって定義される言語のクラスの研究が始められた．最初の PS 完全問題は，NP 完全性の重要性を追求した論文として知られる，Karp[4] によって与えられた．演習問題 11.3.2 に示した問題 — 与えられた正則表現は Σ^* と同等であるかどうかを問う問題 — の PS 完全性は，その論文からの引用である．

限定記号付きブール式の PS 完全性は L. J. Stockmeyer によって証明されたが，論文の形で発表されてはいない．シャノン・スイッチング・ゲームの NP 完全性（演習問題 11.3.3）は文献 [1] による．

素数全体の集合が $\mathcal{NP}$ に属すことは Pratt [9] による．合成数全体の集合が $\mathcal{RP}$ に属すことは，Rabin [10] によって最初に示された．同じ頃，まだ証明されてはいないが一般に正しいと信じられている,「一般化リーマン予想」と呼ばれる予想が正しいと仮定すれば，素数全体の集合は $\mathcal{P}$ に属す，という興味深い結果が証明された（[6]）．

本章で扱われた話題について，さらに進んだ知識を得たい読者のために，いくつか

の書物を紹介しよう．文献 [7] は，ランダム・アルゴリズムについて述べた本であり，素数性判定テストの完全な記述が含まれている．[5] は，モジュラ算術に関するアルゴリズムの出典である．[3] と [8] には，本書では説明しなかったいくつかの計算量のクラスが解説されている．

[1] S. Even and R. E. Tarjan, "A combinatorial problem which is complete for polynomial space," *J. ACM* **23**:4 (1976), pp.710–719.

[2] J. Hartmanis, P. M. Lewis II, and R. E. Stearns, "Hierarchies of memory limited computations," *Proc. Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design* (1965), pp.179–190.

[3] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading MA, 1979.

[4] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations* (R. E. Miller, ed.), Plenum Press, New York, pp.85–104.

[5] D. E. Knuth, *The Art of Computer Programming, Vol. II: Seminumerical Algorithms*, Addison-Wesley, Reading MA, 1997 (third edition).

[6] G. L. Miller, "Riemann's hypothesis and tests for primality," *J. Computer and System Sciences* **13** (1976), pp.300–317.

[7] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, 1995.

[8] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading MA, 1994.

[9] V. R. Pratt, "Every prime has a succinct certificate," *SIAM J. Computing* **4**:3 (1975), pp.214–220.

[10] M. O. Rabin, "Probabilistic algorithms," in *Algorithms and Complexity: Recent Results and New Directions* (J. F. Traub, ed.), pp.21–39, Academic Press, New York, 1976.

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM* **21** (1978), pp.120–126.

[12] W. J. Savitch, "Relationships between deterministic and nondeterministic tape complexities," *J. Computer and System Science* **4**:2 (1970), pp.177–192.

邦訳文献

[3] 野崎昭弘・高橋正子・町田　元・山崎秀記共訳：オートマトン 言語理論 計算論 I, II（サイエンス社，1984，1986）

索　引

あ 行

か 行

さ 行

た 行

な 行

は 行

ま　行

や　行

ら　行

欧　字

記　号

著者略歴

野崎昭弘（のざき あきひろ）

1959年　東京大学理学部数学科卒業
1961年　東京大学大学院修士課程修了
現　在　大妻女子大学教授　理学博士

高橋正子（たかはし まさこ）

1962年　お茶の水女子大学理学部数学科卒業
1972年　ペンシルバニア大学大学院博士課程修了
現　在　国際基督教大学教授
　　　　Ph.D.(ペンシルバニア大学)

町田　元（まちだ はじめ）

1970年　東京大学理学部数学科卒業
1974年　東京大学大学院修士課程修了
現　在　一橋大学教授　博士(数理科学)

山崎秀記（やまざき ひでき）

1972年　東京大学教養学部基礎科学科卒業
1975年　東京大学大学院修士課程修了
現　在　一橋大学教授　理学博士

Information & Computing=4

オートマトン 言語理論 計算論 II [第2版]

1986年 3月10日 ©　初版発行
1998年 3月25日　初版第5刷発行
2003年 8月 5日 ©　第2版第1刷発行

原著者　J. ホップクロフト
　　　　R. モトワニ
　　　　J. ウルマン
訳　者　野崎昭弘
　　　　高橋正子
　　　　町田　元
　　　　山崎秀記

発行者　森平勇三
印刷者　篠倉正信
製本者　金野　明

発行所　**株式会社　サイエンス社**

〒151–0051　東京都渋谷区千駄ヶ谷1丁目3番25号
営業☎(03) 5474–8500 (代) 振替 00170–7–2387
編集☎(03) 5474–8600 (代) FAX ☎(03) 5474–8900

印刷　(株) ディグ　　製本　(株) 積信堂

《検印省略》

ISBN4–7819–1027–0

PRINTED IN JAPAN

サイエンス社のホームページのご案内
http://www.saiensu.co.jp
ご意見・ご要望は
rikei@saiensu.co.jp　まで．